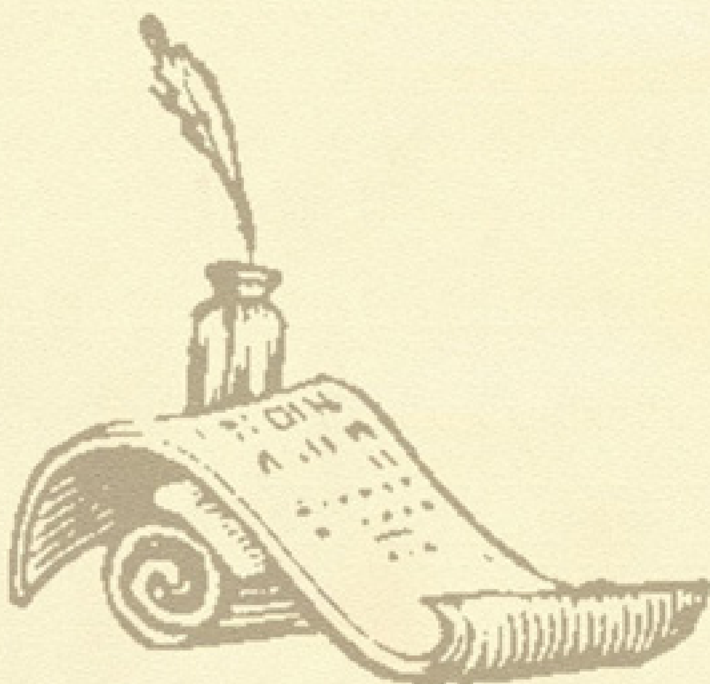


Nicola Bassi

OPEN SOURCE

Analisi di un movimento



APOGEO

SinTesi

Open Source

Autore:
Nicola Bassi

Via Natale Battaglia 12 – 20127 Milano (Italy)
Telefono: 02-28970277 (5 linee r.a.)
Telefax: 02-26116334
Email apogeo@apogeonline.com
U.R.L. http://www.apogeonline.com

Responsabile editoria digitale: Alberto Mari
Copertina: Enrico Marcandalli

A chi grida in silenzio

Indice generale

Ringraziamenti

Abitualmente questa sezione viene dedicata all'espressione di un sincero sentimento di riconoscenza verso congiunti e intimi per fatti quali essere stati messi al mondo, non esserne stati tolti nei momenti di nervosismo, avere ricevuto adeguate sponsorizzazioni per poter godere del diritto all'istruzione sancito dalla Costituzione Italiana e cose così .

Mi assoggetto volentieri alla tradizione e comincio a ringraziare quindi i veri fautori di tutto ciò: papà Lucio e mamma Paola (in ordine alfabetico, NdA). Senza di loro non avrei mai potuto affrontare questa tesi e non solo! Non so se le accademie scientifiche Ve ne saranno grate ma io sicuramente sì . Non farò mai abbastanza per poterVi ringraziare! (Mamma?!? Ma è un modo di dire. . . !).

Grazie anche a Sonia, la mia fidanzata, per il sostegno e l'incoraggiamento fornitomi e soprattutto per la continua magia che mi dona. Grazie!

Grazie al Collegio Crocetta, mia dimora nella maggior parte degli anni da universitario; ai tanti compagni avuti per le partite di calcetto, le bevute, le nottate, i Cinghiali e perché no anche lo studio. Tutto ciò è diventato ormai una parte di me. Se non Vi avessi conosciuto mi sarei laureato anche prima ma mi sarei perso qualcosa.

Grazie dell'ospitalità a Fabrizio e Agata e a Sergio e Barbara, dopo il forzato allontanamento dal Collegio ho sempre potuto contare su di loro per un letto (o divano) e un piatto caldo (vegetariano).

Grazie alla lungimiranza della dirigenza della grossa società presso la quale svolgo la mia opera di consulenza per aver impedito che svolgessi una tesi che sarebbe servita soprattutto a loro. Ho dovuto cambiare argomento della tesi, prendere due mesi di aspettativa, lavorare la notte e i weekend ma mi sono divertito molto di più!

Grazie ai dirigenti della società per cui lavoro, si sono sempre dimostrati disponibili a venirmi incontro per fare in modo che portassi a termine questo lavoro.

Grazie ad Alberto per... boh??? Ma è mio fratello e pareva brutto non citarlo.

Grazie ai colleghi del gruppo PRP: mi hanno consigliato, aiutato e letto stralci della tesi. Hanno reso più sereni questi mesi di duro lavoro.

Grazie a tutti.

Nicola Bassi

Introduzione

L'Open Source ha raggiunto la notorietà in tempi molto recenti, oggi viene menzionato e discusso non solo dalla stampa specializzata ma anche su media più generici come le trasmissioni televisive e radiofoniche. Il fatto che tanta attenzione sia dedicata a un particolare fenomeno legato al mondo dell'informatica e non strettamente connesso con l'utenza popolare, come è il caso Internet, è indice di novità non ristretta all'ambito tecnico, ma legata a un ambito più vasto, l'ambito delle innovazioni culturali.

Eppure l'Open Source non è un fenomeno recente, anzi, l'Open Source è stato il primo modus operandi dell'informatica. Il software nacque come Open Source negli storici laboratori che per primi si occuparono di informatica: i Bell Labs, lo Xerox Park, il IA Lab del MIT, Berkeley. Allora non c'era bisogno di porre distinzioni tra le licenze di software o la distribuzione degli eseguibili piuttosto che dei sorgenti, ciò che veniva creato diventava patrimonio della comunità. Non si trattava di una scelta politica, la libera distribuzione era frutto della constatazione che il software cresce in stabilità, prestazioni, funzionalità se può essere interamente compreso e modificato dai suoi utenti. Il software era un prodotto scientifico, come la matematica e la fisica, e come tale veniva trattato. Così come di un esperimento scientifico si distribuiscono le ipotesi, il procedimento e i risultati, del software si distribuivano l'analisi dei requisiti e il codice sorgente, in modo che tutti potessero valutarne i risultati.

Il software crebbe rapidamente in possibilità di utilizzo interessando il mondo commerciale che vide nei programmi un prodotto manifatturiero su cui esercitare un diritto di proprietà da proteggere con licenze d'uso. Il mercato in rapidissima crescita e ad altissimo reddito attirò i tecnici del software che incominciarono a produrre software sotto il riserbo del segreto industriale e a distribuirlo in forma eseguibile dietro pagamento.

Il software libero, o free software, incominciò a essere inteso come software gratuito e sottintendente la scarsa qualità. Così per circa quindici an-

ni l'attenzione degli utenti è stata rivolta ai produttori commerciali che riuscivano a imporre il proprio prodotto anche a scapito dei contenuti tecnici di questo. Il marketing era importante tanto quanto il software, le battaglie commerciali e legali tra produttori lo testimoniano

Perché si è tornato a parlare di Free Software, o con un termine più moderno, di Open Source? Il fenomeno Open Source è giunto alla ribalta delle cronache grazie a un prodotto di grande impatto sul pubblico: Linux.

Che cos'è Linux? Linux è molte cose insieme: è un sistema operativo completo, Unix compatibile, performante, l'unico oggi in grado di strappare quote di mercato nel settore server a Microsoft; è fornito accompagnato da applicativi di ogni genere; è scalabile dal palmare alla rete di calcolo; è assolutamente gratuito. Questi elementi, i più percettibili ma non gli unici, hanno decretato la riabilitazione del software liberamente, e interamente, distribuibile.

Linux è la punta di un iceberg cresciuto, all'insaputa del mondo commerciale e senza pubblicità, in Internet e gestito da una pletora di appassionati di informatica che programmano per il piacere di programmare: gli hacker, come si definiscono. Invito a non lasciarsi fuorviare dal significato che i media hanno attribuito alla parola hacker, non sono pirati né fuorilegge, sono gli esperti dei sistemi informatici, come verrà documentato più avanti. L'iceberg è costituito dal software liberamente disponibile che, guarda caso, comprende anche programmi che permettono l'utilizzo di Internet: BIND e DNS, per fare un esempio.

Linux rimane comunque il caso più eclatante: nacque nel silenzio della casa di uno studente finlandese, crebbe accudito da una comunità di monaci programmatori la cui dimora è Internet, divenne release e uscì dal convento. Fu screditato dai produttori di sistemi operativi commerciali come un bel giocattolo per gli smanettoni della rete. Dimostrò, release dopo release, un tasso di crescita senza precedenti nel campo dei sistemi operativi, sorpassò nei test di stabilità e di prestazioni i concorrenti diretti Unix, è oggi il primo concorrente, come quote di mercato, di Windows NT.

C'è ancora un particolare da notare: Linus Torvalds, lo studente finlandese, varò il progetto Linux nel 1990.

Come è stato possibile organizzare centinaia di super tecnici sparsi per la Terra senza una struttura preposta, senza fondi e assolutamente senza profitto monetario? Perché tante persone hanno aderito? Come è possibile creare

il kernel di un sistema operativo sul modello Unix contando su contributi di codice volontari?

Linux è figlio di diversi padri: della tribù hacker, dell'idea di Free Software, dell'originalità organizzativa di Linus Torvalds.

Questa tesi ha lo scopo di portare chiarezza sul concetto, sul metodo di produzione, sui prodotti, e sul movimento di programmatori di software cosiddetto "Open Source". Per conseguire lo scopo prefissato si procederà per passi.

In primo luogo definiremo cosa si intende per "Open Source" e tracceremo una chiara mappa atta a classificare i diversi tipi di licenza d'uso del software oggi in circolazione.

Sarà a questo punto evidente la necessità di delineare le origine storiche del movimento che ha portato alla stesura della "Open Source Definition", si descriverà lo stato attuale di tale movimento e si cercherà di tracciarne le prospettive.

Un fenomeno è rilevante dal punto di vista ingegneristico se è riproducibile e controllabile. Nel terzo capitolo affronteremo l'argomento Open Source analizzando, tramite un esperimento realizzato da Eric Raymond, l'effettiva riproducibilità delle condizioni che hanno permesso a Linux, per esempio, di raggiungere determinati risultati. Verrà anche affrontato l'argomento dal punto di vista economico, cioè verranno valutate diverse possibilità, già seguite o possibili, di ricavare profitto distribuendo liberamente il software.

Seguirà l'esposizione di un progetto commerciale per la raccolta e l'analisi di dati medici interamente sviluppato con strumenti Open Source dall'autore.

In ultimo verranno delineati alcuni paralleli di carattere generale tra il movimento Open Source e i movimenti di innovazione culturale appartenenti al mondo occidentale, dalle analogie riscontrate si proporrà una chiosa al "Progetto Freeware" elaborato dal Prof. Angelo Meo (Politecnico di Torino) [1].

Nelle appendici verrà riportata prima una raccolta di licenze omologate come "OSI Certfied" e di seguito l'esposizione dei prodotti più noti rilasciati come software libero. L'elenco non è assolutamente esaustivo, si intende solo portare degli esempi pratici delle potenzialità del software Open Source. Si noterà che dall'elenco manca Linux, per l'unica ragione che viene trattato esaurientemente nel corso della tesi.

1 Il software Open Source

1.1 Introduzione

In termini legali il software Open Source è software distribuito con una licenza che ne consente la libera distribuzione in forma sorgente, e conferisce la possibilità all'utente di poter modificare il programma originario e di poter distribuire la versione modificata.

Questo capitolo intende definire con chiarezza l'entità di tali licenze e dei canoni che bisogna rispettare per definire un software Open Source. Per raggiungere tale scopo è necessario chiarire quali altri tipi di software, classificati per licenza (o assenza della licenza), esistono e in che relazione sono.

La classificazione è utile anche per distinguere ciò che viene definito come Free Software dall'Open Source. La differenza è molto sottile e nasce dal problema di diffondere il concetto di software libero presso il mondo commerciale. Infatti il termine "free" in inglese significa sia libero che gratis, questa ambivalenza ha generato spesso confusione tra chi si avvicinava, dal punto di vista imprenditoriale, al mondo del Free Software. Per il momento è sufficiente chiarire che sono due insiemi quasi interamente sovrapposti e che "Free software" è l'idea che sta alla base e "Open Source" è il tentativo di renderla accessibile ai produttori del mondo commerciale. La difficoltà è stata garantire il diritto di libera distribuzione ai programmatori insieme al diritto di poter ottenere un guadagno per gli investitori. Motivo per cui si è rischiata una profonda frattura all'interno della comunità hacker, frattura scongiurata dalla nascita della "Open Source Initiative" e dalla buona disposizione delle personalità maggiormente coinvolte nello scontro tra "puristi" e "possibilisti": Richard Stallman, fondatore della "Free Software Foundation", e Eric Raymond, cofondatore della "Open Source Initiative".

1.2 Una prima generale classificazione

Vengono qui presentate le diverse tipologie di software classificate per licenza di distribuzione, l'elenco è utile per tracciare una mappa del software per definire il più precisamente possibile l'argomento della tesi.

1.2.1 Free Software

Free Software è software che viene distribuito accompagnato dal permesso per chiunque di essere usato, copiato e distribuito, sia integralmente sia modificato, sia gratis che per un compenso.

In particolare, questo significa che il codice sorgente deve essere disponibile al fine di garantire la reale possibilità di modificare il programma.

La Free Software Foundation (FSF) [2], fondata da Richard Stallman nel 1984, si occupa di patrocinare il Free Software.

Per evitare che il software rilasciato potesse essere facilmente cooptato dai produttori commerciali venne rilasciata una licenza, la General Purpose License (GPL), che ancora oggi è punto di riferimento per tutto ciò che è Free Software.

I termini della licenza non permettono in pratica nessuna commercializzazione del software. Permettono la possibilità di introito derivante dalla vendita del supporto fisico del programma o dalle varie forme di consulenza sul programma stesso.

La FSF nasce da idee libertarie che vanno oltre il campo di applicazione del software. Tali idee sono state mal recepite dalla comunità dei produttori commerciali che infatti non hanno mai aderito alle iniziative di Stallman.

Gran parte dell'incomprensione è dovuta principalmente all'uso del termine free. Free in inglese indica sia ciò che è libero, senza vincoli, sia ciò che è gratuito. Benché Stallman si sforzasse di chiarire l'equivoco con il motto "Free speach, not free beer!", intendendo riferirsi alla libertà di lavorare sul codice e non alla libertà di potersi appropriare del codice altrui, l'idea di Free Software venne vista come un tentativo di attentare alla proprietà privata e ai diritti di copyright.

1.2.2 Open Source

Per sanare l'incomprensione tra i sostenitori di una libera distribuzione del software sorgente e le compagnie che fanno della proprietà del codice la loro fonte di reddito, si rese necessario chiarire, aldilà dei termini usati la reale portata, e i conseguenti vantaggi, della distribuzione del codice sorgente. Quando la Netscape decise di lanciare il progetto Mozilla, distribuendo i sorgenti del browser "Netscape Navigator", divenne palese l'esigenza di coprire gli interessi commerciali della libera impresa insieme alla garanzia, per la comunità di sviluppatori, del libero utilizzo del codice.

Nacque la "Open Source Initiative" (OSI) [3] nel 1997, un'organizzazione non a scopo di lucro, avente come referente guida Eric Raymond. La OSI registrò il marchio "OSI Certified", per garantire agli sviluppatori la possibilità legale del riutilizzo e della distribuzione del codice proprietario, e rilasciò la "Open Source Definition", un insieme di regole che le licenze che accompagnano il software devono rispettare per potersi dotare del marchio "OSI Certified".

1.2.3 Public Domain Software

Il software di pubblico dominio è software che non ha il copyright. È un caso particolare di Free Software non coperto da nessun tipo di licenza: questo significa che alcune copie o versioni modificate possono non essere libere.

A volte si usa il termine Dominio Pubblico per indicare il software libero o disponibile senza spese. Per la verità "Dominio Pubblico" è un termine legale che significa, per la precisione, senza copyright.

1.2.4 Copylefted Software

Il termine "Copyleft" fu coniato ironicamente da Richard Stallman.

Il modo più semplice per rendere un programma free software è quello di dichiararlo come Public Domain. Il problema consiste nel fatto che versioni del programma originario possono diventare versioni proprietarie. Per impedire questo Stallman suggerisce di rendere il software "Copylefted", ossia di usare, per esempio, il copyright della General Purpose License per garantire che anche le copie modificate dovranno rispettare i termini di distribuzione dell'autore.

1.2.5 Free Software non-copylefted

È il software che l'autore distribuisce con il permesso di ridistribuzione e di modifica. L'autore permette anche l'aggiunta di ulteriori restrizioni ai distributori. È il caso di X-Window, la piattaforma Unix per interfacce grafiche.

Sviluppata al MIT fu rilasciata assolutamente libera, importando unicamente ai suoi autori che venisse usata. Accadde così che ogni società che la modificava per renderla compatibile ad una particolare piattaforma diventava proprietaria di quella particolare versione del codice. Di conseguenza l'utente finale di un prodotto nato come Free Software ha sempre solo usato versioni proprietarie.

1.2.6 Semi-free software

Semi-free software è software che non è libero, ma è rilasciato con il permesso di poter essere usato, modificato e distribuito senza fini di lucro. In genere è software liberamente utilizzabile da amatori e scuole o associazioni. Non viene considerato free software perché il free software deve poter essere utilizzato (e ridistribuito) da tutti, compreso chi ne fa la sua fonte di guadagno.

1.2.7 Software Proprietario

Il software proprietario è software che non è libero né semilibero. Il suo uso, la distribuzione, la modifica sono proibite o richiedono un permesso o sono ristrette sotto condizioni tali da essere di fatto impedite.

1.2.8 Freeware

Il termine "freeware" non delinea univocamente un certo tipo di software. In genere viene usato per pacchetti per i quali è permessa la distribuzione ma non le modifiche (il loro codice sorgente non è disponibile). Da notare che la condizione prima del free software è la possibilità di modificare il sorgente.

1.2.9 Shareware

Shareware è software del quale è permessa la distribuzione, ma in caso di utilizzo è richiesto il pagamento di una licenza d'uso.

Lo shareware non è free software:

- In genere il codice sorgente non è disponibile
- È necessario, per l'utilizzo, il pagamento di una licenza

1.2.10 Software Commerciale

Il software commerciale è software che viene sviluppato da una compagnia che desidera ottenere una rendita dall'uso di quel software. Software commerciale e proprietario non sono la stessa cosa! La maggior parte del software commerciale è proprietario, ma esiste free software commerciale ed esiste software proprietario non commerciale e non libero.

Ad esempio ci sono società che producono software e lo distribuiscono come free software ma vendono il supporto e la consulenza agli utilizzatori di quel software. Questo è software commerciale, ma non proprietario.

La classificazione precedente tradotta in mappa:

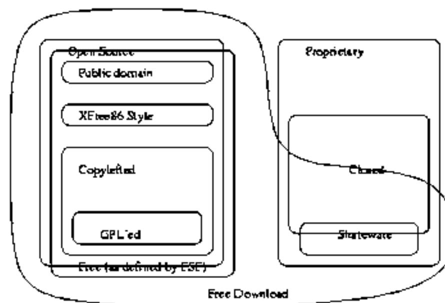


Figura 1.1: Mappa delle licenze software

1.3 La General Purpose License

La General Purpose License (GPL) fu redatta dalla Free Software Foundation (FSF) nel 1989, e poi rivista nel 1991, con lo scopo di proteggere legalmente

lo sviluppo di software libero. Il pericolo principale era quello di regalare, come sempre avveniva, sorgenti altamente performanti e/o innovativi alle società che facevano della proprietà del software un business.

Il software rilasciato come Pubblico Dominio è facile preda: basta apporre una piccola modifica, l'estensione di una funzionalità per rivendicare il possesso della versione modificata.

Richard Stallman, leader della FSF, si accorse che era necessario qualcosa in più per proteggere il patrimonio di codice libero che sgorgava dalla rete e soprattutto per assicurare che codice nato come libero restasse libero. Occorreva una licenza che consentisse la libera distribuzione del programma originale e delle copie modificate o, per dirla alla Stallman: "Serviva un copyright che garantisse il copyleft!".

1.3.1 Cosa era necessario proteggere?

O meglio: in cosa consiste la libertà relativa al software? Il Free Software non è software relativo a particolari applicazioni né sistemi, e neanche free si riferisce a una qualche specifica tecnica di progetto.

Free Software è relativo alla libertà dell'utente di eseguire, copiare, distribuire, studiare, cambiare e migliorare il software.

Più precisamente si riferisce a quattro tipi di libertà capitali per l'utente di software:

- La libertà di eseguire il programma, a qualsiasi scopo (Libertà n. 0).
- La libertà di studiare come il programma lavora, e adattarlo alle proprie necessità (Libertà n. 1).
- La libertà di distribuire copie a chi ne ha bisogno (Libertà n.2).
- La libertà di migliorare il programma e rilasciare i propri miglioramenti al pubblico, in modo che l'intera comunità ne benefici (Libertà n.3).

Ovviamente le libertà n. 1 e n. 3 sottintendono la possibilità di accedere al codice sorgente.

Per la FSF un programma è libero se gli utenti hanno queste libertà. Quindi si dovrebbe essere liberi di ridistribuire copie, con o senza modifiche, sia

gratis o per un compenso per il supporto, a chiunque e ovunque. Essere liberi di fare queste cose significa (inoltre) che non è necessario chiedere un permesso o pagare per averlo.

Per avere la libertà di porre modifiche e di pubblicare le versioni migliorate è necessario avere l'accesso al codice sorgente del programma. Di conseguenza l'accessibilità al codice sorgente è una condizione necessaria al Free Software.

Perché tali libertà siano reali devono essere irrevocabili finché non viene commesso dolo (i.e.: per casi eccezionali); se lo sviluppatore del software avesse il potere di revocare la licenza, anche senza motivo, il software non sarebbe libero. È così necessario garantire che tali libertà siano rispettate lungo tutto l'arco delle distribuzioni e delle modifiche del codice.

Finché vengono mantenute le libertà fondamentali è lecito parlare di Free Software. Il copyleft è una regola che non permette di aggiungere restrizioni alle quattro libertà capitali, di conseguenza può sembrare esso stesso un limite alla libertà. Ma questa regola non entra in conflitto con quelle libertà, piuttosto le protegge da restrizioni successive.

Per la FSF sono accettabili regole su come creare un pacchetto con le versioni modificate, se non limitano la possibilità stessa di rilasciare versioni modificate. Regole come "se rendi il programma disponibile nel modo X devi anche renderlo disponibile nel modo Y" sono ritenute accettabili, sempre che rispettino la condizione di non limitare il rilascio delle nuove versioni. L'importante è che tali regole permettano ancora la libertà di rilasciare o meno le versioni modificate.

È interessante notare l'orientamento della FSF nei confronti delle leggi di embargo economico e di beni. Nel caso di leggi governative sull'esportazione e di sanzioni commerciali limitanti la libertà nella distribuzione internazionale delle copie di un programma la FSF è perentoria: "Gli sviluppatori di software non hanno il potere di eliminare o scavalcare tali restrizioni, ma ciò che possono e devono fare è rifiutare di imporre quelle leggi sulla licenza d'uso del software. In questo modo le restrizioni non intaccheranno le attività e la gente fuori dalla giurisdizione di quei governi."

Insomma la libertà del software non deve venire arrestata dalle frontiere!

1.3.2 Il testo della General Purpose License

La GPL sancisce i principi sopra esposti. Qui verrà presentata una traduzione in italiano [4] che non ha valore legale. La versione ufficiale, in lingua originale, sarà presentata in appendice.

Come apparirà subito evidente il riferimento al progetto GNU è onnipresente. Questo è dovuto al fatto che la FSF è figlia del progetto GNU. Nel capitolo dedicato alla storia dell'Open Source sarà tutto più chiaro. Per il momento basterà notare che il progetto GNU (acronimo recursivo per GNU's Not Unix) intende costruire un ambiente completo (dal sistema operativo agli applicativi per il desktop) interamente Unix compatibile e interamente Free Software. La GPL nacque in seno a GNU per proteggerne lo sviluppo dal cannibalismo del mercato.

Disclaimer

Questa è una traduzione italiana non ufficiale della Licenza Pubblica Generale GNU. Non è pubblicata dalla Free Software Foundation e non ha valore legale nell'esprimere i termini di distribuzione del software che usa la licenza GPL. Solo la versione originale in inglese della licenza ha valore legale. Ad ogni modo, speriamo che questa traduzione aiuti le persone di lingua italiana a capire meglio il significato della licenza GPL.

This is an unofficial translation of the GNU General Public License into Italian. It was not published by the Free Software Foundation, and does not legally state the distribution terms for software that uses the GNU GPL - only the original English text of the GNU GPL does that. However, we hope that this translation will help Italian speakers understand the GNU GPL better.

LICENZA PUBBLICA GENERICA (GPL) DEL PROGETTO GNU

Versione 2, Giugno 1991

Copyright (C) 1989, 1991 Free Software Foundation, Inc. 675 Mass Ave, Cambridge, MA 02139, USA

Preambolo

Le licenze per la maggioranza dei programmi hanno lo scopo di togliere all'utente la libertà di condividere e di modificare il programma stesso. Al

contrario, la Licenza Pubblica Generica GNU è intesa a garantire la libertà di condividere e modificare il free software, al fine di assicurare che i programmi siano “liberi” per tutti i loro utenti. Questa Licenza si applica alla maggioranza dei programmi della Free Software Foundation e ad ogni altro programma i cui autori hanno scelto questa Licenza. Alcuni altri programmi della Free Software Foundation sono invece coperti dalla Licenza Pubblica Generica per Librerie. Chiunque puo‘ usare questa Licenza per i propri programmi.

Quando si parla di “free software”, ci si riferisce alla libertà, non al prezzo. Le nostre Licenze (la GPL e la LGPL) sono progettate per assicurarsi che ciascuno abbia la libertà di distribuire copie del free software (e farsi pagare per questo, se vuole), che ciascuno riceva il codice sorgente o che lo possa ottenere se lo desidera, che ciascuno possa modificare il programma o usarne delle parti in nuovi programmi “liberi” e che ciascuno sappia di potere fare queste cose.

Per proteggere i diritti dell’utente, abbiamo bisogno di creare delle restrizioni che vietino a chiunque di negare questi diritti o di chiedere di rinunciarvi. Queste restrizioni si traducono in certe responsabilità per chi distribuisce copie del software e per chi lo modifica.

Per esempio, chi distribuisce copie di un Programma coperto da GPL, sia gratis sia in cambio di un compenso, deve dare ai destinatari tutti i diritti che ha ricevuto. Deve anche assicurarsi che i destinatari ricevano o possano ricevere il codice sorgente. E deve mostrare loro queste condizioni di Licenza, in modo che conoscano i loro diritti.

Proteggiamo i diritti dell’utente in due modi: (1) proteggendo il software con un copyright, e (2) offrendo una Licenza che offre il permesso legale di copiare, distribuire e/o modificare il Programma.

Infine, per proteggere ogni autore e noi stessi, vogliamo assicurarci che ognuno capisca che non ci sono garanzie per i programmi coperti da GPL. Se il Programma viene modificato da qualcun altro e ridistribuito, vogliamo che gli acquirenti sappiano che ciò che hanno non è l’originale, in modo che ogni problema introdotto da altri non si rifletta sulla reputazione degli autori originari.

Infine, ogni programma libero è costantemente minacciato dai brevetti sui programmi. Vogliamo evitare il pericolo che chi ridistribuisce un Programma libero ottenga brevetti personali, rendendo in questo modo il Programma una cosa di sua proprietà. Per prevenire questo, abbiamo chiarito che ogni prodot-

to brevettato debba essere distribuito per il libero uso da parte di chiunque, o non distribuito affatto.

Seguono i termini e le condizioni precisi per la copia, la distribuzione e la modifica.

Licenza Pubblica Generica GNU: termini e condizioni per la copia, la distribuzione e la modifica

0. Questa Licenza si applica a ogni Programma o altra opera che contenga una nota da parte del detentore del copyright che dica che tale opera può essere distribuita sotto i termini di questa Licenza Pubblica Generica. Il termine "Programma" nel seguito indica ognuno di questi programmi o lavori, e l'espressione "lavoro basato sul Programma" indica sia il Programma sia ogni opera considerata "derivata" in base alla legge sul Copyright: cioè un lavoro contenente il programma o una porzione di esso, sia letteralmente sia modificato e/o tradotto in un'altra lingua; da qui in avanti, la traduzione è in ogni caso considerata una "modifica". Vengono ora elencati i diritti dei detentori di licenza.

   Attività diverse dalla copiatura, distribuzione e modifica non sono coperte da questa Licenza e sono al di fuori della sua influenza. L'atto di eseguire il programma non viene limitato, e l'output del programma è coperto da questa Licenza solo se il suo contenuto costituisce un lavoro basato sul Programma (indipendentemente dal fatto che sia stato creato eseguendo il Programma). In base alla natura del Programma il suo output può essere o meno coperto da questa Licenza.

1. È lecito copiare e distribuire copie letterali del codice sorgente del Programma così come viene ricevuto, con qualsiasi mezzo, a condizione che venga riprodotta chiaramente su ogni copia una appropriata nota di copyright e di assenza di garanzia; che si mantengano intatti tutti i riferimenti a questa Licenza e all'assenza di ogni garanzia; che si dia a ogni altro destinatario del Programma una copia di questa Licenza insieme al Programma.

   È possibile richiedere un pagamento per il trasferimento fisico di una copia del Programma, è anche possibile a propria discrezione richiedere un pagamento in cambio di una copertura assicurativa.

2. È lecito modificare la propria copia o copie del Programma, o parte di esso, creando perciò un lavoro basato sul Programma, e copiare o distribuire queste modifiche e questi lavori sotto i termini del precedente punto 1, a patto che anche tutte queste condizioni vengano soddisfatte:

    a) Bisogna indicare chiaramente nei file che si tratta di copie modificate e la data di ogni modifica.
    b) Bisogna fare in modo che ogni lavoro distribuito o pubblicato, che in parte o nella sua totalità derivi dal Programma o da parti di esso, sia globalmente utilizzabile da terze parti secondo le condizioni di questa licenza.
    c) Se di solito il programma modificato legge comandi interattivamente quando eseguito, bisogna fare in modo che all'inizio dell'esecuzione interattiva usuale, stampi un messaggio contenente una appropriata nota di copyright e di assenza di garanzia (oppure che specifichi il tipo di garanzia che si offre). Il messaggio deve inoltre specificare agli utenti che possono ridistribuire il programma nelle condizioni qui descritte e deve indicare come reperire questa licenza. Se però il programma di partenza è interattivo ma normalmente non stampa tale messaggio, non occorre che un lavoro derivato lo stampi.

    Questi requisiti si applicano al lavoro modificato nel suo complesso. Se sussistono parti identificabili del lavoro modificato che non siano derivate dal Programma e che possono essere ragionevolmente considerate lavori indipendenti, allora questa Licenza e i suoi termini non si applicano a queste parti quando vengono distribuite separatamente. Se però queste parti vengono distribuite all'interno di un prodotto che è un lavoro basato sul Programma, la distribuzione di questo prodotto nel suo complesso deve avvenire nei termini di questa Licenza, le cui norme nei confronti di altri utenti si estendono a tutto il prodotto, e quindi ad ogni sua parte, chiunque ne sia l'autore.

    Sia chiaro che non è nelle intenzioni di questa sezione accampare diritti su lavori scritti interamente da altri, l'intento è piuttosto quello di esercitare il diritto di controllare la distribuzione di lavori derivati o dal Programma o contenenti esso.

Inoltre, se il Programma o un lavoro derivato da esso viene aggregato ad un altro lavoro non derivato dal Programma su di un mezzo di immagazzinamento o di distribuzione, il lavoro non derivato non deve essere coperto da questa licenza.

3. È lecito copiare e distribuire il Programma (o un lavoro basato su di esso, come espresso al punto 2) sotto forma di codice oggetto o eseguibile sotto i termini dei precedenti punti 1 e 2, a patto che si applichi una delle seguenti condizioni:

   a) Il Programma sia corredato dal codice sorgente completo, in una forma leggibile dal calcolatore e tale sorgente deve essere fornito secondo le regole dei precedenti punti 1 e 2 su di un mezzo comunemente usato per lo scambio di programmi.
   b) Il Programma sia accompagnato da un'offerta scritta, valida per almeno tre anni, di fornire a chiunque ne faccia richiesta una copia completa del codice sorgente, in una forma leggibile dal calcolatore, in cambio di un compenso non superiore al costo del trasferimento fisico di tale copia, che deve essere fornita secondo le regole dei precedenti punti 1 e 2 su di un mezzo comunemente usato per lo scambio di programmi.
   c) Il Programma sia accompagnato dalle informazioni che sono state ricevute riguardo alla possibilità di avere il codice sorgente. Questa alternativa è permessa solo in caso di distribuzioni non commerciali e solo se il programma è stato ricevuto sotto forma di codice oggetto o eseguibile in accordo al precedente punto B.

   Per "codice sorgente completo" di un lavoro si intende la forma preferenziale usata per modificare un lavoro. Per un programma eseguibile, "codice sorgente completo" significa tutto il codice sorgente di tutti i moduli in esso contenuti, più ogni file associato che definisca le interfacce esterne del programma, più gli script usati per controllare la compilazione e l'installazione dell'eseguibile. In ogni caso non e' necessario che il codice sorgente fornito includa nulla che sia normalmente distribuito (in forma sorgente o in formato binario) con i principali componenti del sistema operativo sotto cui viene eseguito il Pro-

gramma (compilatore, kernel, e così via), a meno che tali componenti accompagnino l'eseguibile.

Se la distribuzione dell'eseguibile o del codice oggetto è effettuata indicando un luogo dal quale sia possibile copiarlo, permettere la copia del codice sorgente dallo stesso luogo è considerata una valida forma di distribuzione del codice sorgente, anche se copiare il sorgente è facoltativo per l'acquirente.

4. Non è lecito copiare, modificare, sublicenziare, o distribuire il Programma in modi diversi da quelli espressamente previsti da questa Licenza. Ogni tentativo di copiare, modificare, sublicenziare o distribuire il Programma non è autorizzato, e farà terminare automaticamente i diritti garantiti da questa Licenza. D'altra parte ogni acquirente che abbia ricevuto copie, o diritti, coperti da questa Licenza da parte di persone che violano la Licenza come qui indicato non vedranno invalidare la loro Licenza, purché si comportino conformemente ad essa.

5. L'acquirente non è obbligato ad accettare questa Licenza, poiché non l'ha firmata. D'altra parte nessun altro documento garantisce il permesso di modificare o distribuire il Programma o i lavori derivati da esso. Queste azioni sono proibite dalla legge per chi non accetta questa Licenza; perciò, modificando o distribuendo il Programma o un lavoro basato sul programma, si indica nel fare ciò l'accettazione di questa Licenza e quindi di tutti i suoi termini e le condizioni poste sulla copia, la distribuzione e la modifica del Programma o di lavori basati su di esso.

6. Ogni volta che il Programma o un lavoro basato su di esso vengono distribuiti, l'acquirente riceve automaticamente una licenza d'uso da parte del licenziatario originale. Tale licenza regola la copia, la distribuzione e la modifica del Programma secondo questi termini e queste condizioni. Non è lecito imporre restrizioni ulteriori all'acquirente nel suo esercizio dei diritti qui garantiti. Chi distribuisce programmi coperti da questa Licenza non è comunque responsabile per la conformità alla Licenza da parte di terze parti.

7. Se, come conseguenza del giudizio di una corte, o di una imputazione per la violazione di un brevetto o per ogni altra ragione (anche non

relativa a questioni di brevetti), vengono imposte condizioni che contraddicono le condizioni di questa licenza, che queste condizioni siano dettate dalla corte, da accordi tra le parti o altro, queste condizioni non esimono nessuno dall'osservazione di questa Licenza. Se non è possibile distribuire un prodotto in un modo che soddisfi simultaneamente gli obblighi dettati da questa Licenza e altri obblighi pertinenti, il prodotto non può essere affatto distribuito. Per esempio, se un brevetto non permettesse a tutti quelli che lo ricevono di ridistribuire il Programma senza obbligare al pagamento di diritti, allora l'unico modo per soddisfare contemporaneamente il brevetto e questa Licenza è di non distribuire affatto il Programma.

Se parti di questo punto sono ritenute non valide o inapplicabili per qualsiasi circostanza, deve comunque essere applicata l'idea espressa da questo punto; in ogni altra circostanza invece deve essere applicato il punto 7 nel suo complesso.

Non è nello scopo di questo punto indurre gli utenti ad infrangere alcun brevetto né ogni altra rivendicazione di diritti di proprietà, né di contestare la validità di alcuna di queste rivendicazioni; lo scopo di questo punto è solo quello di proteggere l'integrità del sistema di distribuzione dei programmi liberi, che viene realizzato tramite l'uso della licenza pubblica. Molte persone hanno contribuito generosamente alla vasta gamma di programmi distribuiti attraverso questo sistema, basandosi sull'applicazione fedele di tale sistema. L'autore/donatore può decidere di sua volontà se preferisce distribuire il software avvalendosi di altri sistemi, e l'acquirente non può imporre la scelta del sistema di distribuzione.

Questo punto serve a rendere il più chiaro possibile ciò che crediamo sia una conseguenza del resto di questa Licenza.

8. Se in alcuni paesi la distribuzione e/o l'uso del Programma sono limitati da brevetto o dall'uso di interfacce coperte da copyright, il detentore del copyright originale che pone il Programma sotto questa Licenza può aggiungere limiti geografici espliciti alla distribuzione, per escludere questi paesi dalla distribuzione stessa, in modo che il programma possa essere distribuito solo nei paesi non esclusi da questa regola. In questo

caso i limiti geografici sono inclusi in questa Licenza e ne fanno parte a tutti gli effetti.

9. All'occorrenza la Free Software Foundation puo' pubblicare revisioni o nuove versioni di questa Licenza Pubblica Generica. Tali nuove versioni saranno simili a questa nello spirito, ma potranno differire nei dettagli al fine di coprire nuovi problemi e nuove situazioni.

   Ad ogni versione viene dato un numero identificativo. Se il Programma asserisce di essere coperto da una particolare versione di questa Licenza e "da ogni versione successiva", l'acquirente può scegliere se seguire le condizioni della versione specificata o di una successiva. Se il Programma non specifica quale versione di questa Licenza deve applicarsi, l'acquirente può scegliere una qualsiasi versione tra quelle pubblicate dalla Free Software Foundation.

10. Se si desidera incorporare parti del Programma in altri programmi liberi le cui condizioni di distribuzione differiscano da queste, è possibile scrivere all'autore del Programma per chiederne l'autorizzazione. Per il software il cui copyright è detenuto dalla Free Software Foundation, si scriva alla Free Software Foundation; talvolta facciamo eccezioni alle regole di questa Licenza. La nostra decisione sarà guidata da due scopi: preservare la libertà di tutti i prodotti derivati dal nostro free software e promuovere la condivisione e il riutilizzo del software in generale.

NON C'E' GARANZIA

11. POICHE' IL PROGRAMMA E' CONCESSO IN USO GRATUITAMENTE, NON C'E' GARANZIA PER IL PROGRAMMA, NEI LIMITI PERMESSI DALLE VIGENTI LEGGI. SE NON INDICATO DIVERSAMENTE PER ISCRITTO, IL DETENTORE DEL COPYRIGHT E LE ALTRE PARTI FORNISCONO IL PROGRAMMA "COSI' COM'E"', SENZA ALCUN TIPO DI GARANZIA, NE' ESPLICITA NE' IMPLICITA; CIO' COMPRENDE, SENZA LIMITARSI A QUESTO, LA GARANZIA IMPLICITA DI COMMERCIABILITA' E UTILIZZABILITA' PER UN PARTICOLARE SCOPO. L'INTERO RISCHIO CONCERNENTE LA QUALITA' E LE PRESTAZIONI DEL PROGRAMMA E' DELL'ACQUIRENTE. SE IL PRO-

GRAMMA DOVESSE RIVELARSI DIFETTOSO, L'ACQUIRENTE SI ASSUME IL COSTO DI OGNI MANUTENZIONE, RIPARAZIONE O CORREZIONE NECESSARIA.

12. NE' IL DETENTORE DEL COPYRIGHT NE' ALTRE PARTI CHE POSSONO MODIFICARE O RIDISTRIBUIRE IL PROGRAMMA COME PERMESSO IN QUESTA LICENZA SONO RESPONSABILI PER DANNI NEI CONFRONTI DELL'ACQUIRENTE, A MENO CHE QUESTO NON SIA RICHIESTO DALLE LEGGI VIGENTI O APPAIA IN UN ACCORDO SCRITTO. SONO INCLUSI DANNI GENERICI, SPECIALI O INCIDENTALI, COME PURE I DANNI CHE CONSEGUONO DALL'USO O DALL'IMPOSSIBILITA' DI USARE IL PROGRAMMA; CIO' COMPRENDE, SENZA LIMITARSI A QUESTO, LA PERDITA DI DATI, LA CORRUZIONE DEI DATI, LE PERDITE SOSTENUTE DALL'ACQUIRENTE O DA TERZE PARTI E L'INABILITA' DEL PROGRAMMA A LAVORARE INSIEME AD ALTRI PROGRAMMI, ANCHE SE IL DETENTORE O ALTRE PARTI SONO STATE AVVISATE DELLA POSSIBILITA' DI QUESTI DANNI.

FINE DEI TERMINI E DELLE CONDIZIONI

Appendice: come applicare questi termini ai nuovi programmi

Se si sviluppa un nuovo programma e lo si vuole rendere della maggiore utilità possibile per il pubblico, la cosa migliore da fare è rendere tale programma free software, cosicché ciascuno possa ridistribuirlo e modificarlo sotto questi termini.

Per fare questo, si inserisca nel programma la nota sotto indicata. La cosa migliore da fare è mettere la nota all'inizio di ogni file sorgente, per chiarire nel modo più efficiente possibile l'assenza di garanzia; ogni file dovrebbe contenere almeno la nota di copyright e l'indicazione di dove trovare l'intera nota.

<una riga per dire in breve il nome del programma e cosa fa>

Copyright (C) 19aa <nome dell'autore>

Questo programma è free software; è lecito ridistribuirlo e/o modificarlo secondo i termini della Licenza Pubblica Generica GNU

> come è pubblicata dalla Free Software Foundation; o la versione 2 della licenza o (a propria scelta) una versione successiva.
>
> Questo programma è distribuito nella speranza che sia utile, ma SENZA ALCUNA GARANZIA; senza neppure la garanzia implicita di NEGOZIABILITA‘ o di APPLICABILITA‘ PER UN PARTICOLARE SCOPO. Si veda la Licenza Pubblica Generica GNU per avere maggiori dettagli.
>
> Ognuno dovrebbe avere ricevuto una copia della Licenza Pubblica Generica GNU insieme a questo programma; in caso contrario, si scriva alla Free Software Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, Stati Uniti.

Si aggiungano anche informazioni su come si può essere contattati tramite posta elettronica e cartacea.

Se il programma è interattivo, si faccia in modo che stampi una breve nota simile a questa quando viene usato interattivamente:

> Orcaloca versione 69, Copyright (C) 19aa <nome dell’autore>
>
> Orcaloca non ha ALCUNA GARANZIA; per i dettagli si digiti ‘show g’. Questo è free software, e ognuno è libero di ridistribuirlo sotto certe condizioni; si digiti ‘show c’ per dettagli.

Gli ipotetici comandi “show g” e “show c” mostreranno le parti appropriate della Licenza Pubblica Generica. Chiaramente, i comandi usati possono essere chiamati diversamente da “show g” e “show c” e possono anche essere selezionati con il mouse o attraverso un menu‘; in qualunque modo pertinente al programma.

Se necessario, si dovrebbe anche far firmare al proprio datore di lavoro (se si lavora come programmatore) o alla propria scuola, se si è studente, una “rinuncia al copyright” per il programma. Ecco un esempio con nomi fittizi:

> Yoyodinamica SPA rinuncia con questo documento ad ogni interesse al copyright del programma “Orcaloca” (che svolge dei passi di compilazione) scritto da Giovanni Smanettone.
>
> <firma di Primo Tizio>, 1 April 1999 Primo Tizio, Presidente

I programmi coperti da questa Licenza Pubblica Generica non possono essere incorporati all'interno di programmi proprietari. Se il proprio programma è una libreria di funzioni, può essere più utile permettere di collegare applicazioni proprietarie alla libreria. Se si ha questa intenzione consigliamo di usare la Licenza Generica Pubblica GNU per Librerie (LGPL) al posto di questa Licenza.

1.3.3 Alcune considerazioni sulla GPL

In che modo la GPL garantisce le libertà fondamentali dell'utente software? Specificando su cosa si estende la licenza.

L'articolo 0 esplicita che la licenza copre tutto ciò che è relativo al programma, le modifiche in primis e se è possibile anche l'output.

Garantendo il diritto di copia e obbligando a distribuire sempre il software con copia della licenza (Art.1). Garantendo il diritto di paternità al creatore.

Ad esempio obbliga (Art.2/a) chi modifica un programma a segnalare il fatto mediante il codice versione, una data, il nominativo dell'autore della modifica con lo scopo di identificare sempre il responsabile del codice. In genere esiste una sezione, sul sorgente o su un file apposito, con la storia delle modifiche e dei loro autori. Questo permette una equa distribuzione di meriti e responsabilità. Se il programma viene modificato ed esteso con parti di codice ragionevolmente indipendenti tali parti non devono sottostare la GPL solo fintantoché non vengono distribuite in un lavoro basato sul programma originario. Quindi in realtà non ci sono speranze di uscire dai canoni di questa licenza. È per questo che è stata dichiarata troppo restrittiva da parte dei produttori commerciali e di conseguenza fu scarsamente adottata.

La GPL stabilisce altresì che deve accompagnare sempre il software (Art.6) e che deve essere applicata integralmente (Art.7), se un qualche tribunale imponesse condizioni sulla distribuzione non conformi alla licenza il programma diviene immediatamente non distribuibile nella giurisdizione di quel tribunale.

È possibile inserire nella GPL l'elenco di Stati in cui quel software non può venire distribuito perché già coperto da altri copyright o brevetti (Art.8).

Gli articoli 11 e 12 svincolano i programmatori e/o distributori da qualsiasi responsabilità derivante da un malfunzionamento del programma. Poiché si tratta di distribuzione gratuita e soprattutto poiché non si ha nessun mezzo per

sapere a chi va finire il software, questi articoli (che come avrete notato sono scritti in stampatello) proteggono in maniera assoluta gli autori da qualsiasi rivalsa. Può sembrare un limite strutturale nello sviluppo del Free Software ma, a parte il fatto che anche il software proprietario è generalmente accompagnato da clausole contrattuali di questo tipo, in realtà permette alle società di sviluppo una fonte di reddito dal proprio lavoro. Come? Non è contro l’etica del Free Software stipulare contratti di assistenza con clienti che hanno bisogno di particolari garanzie. Chi meglio conosce un software se non gli autori stessi? È questo in fondo il business della maggior parte delle società che sviluppano Free Software. Tratteremo nel capitolo 3 casi di business non basati sulla proprietà del software.

La GPL ha il vantaggio di essere chiara e inequivocabile nelle sue clausole. Protegge gli autori e i distributori e garantisce che il loro lavoro continui a evolversi nelle condizioni da loro dettate.

Quella che è la grandezza della GPL è anche un suo limite. Come già accennato la licenza non incontrò le simpatie dei produttori decisi a scommettere sul Free Software. Le maggiori critiche erano rivolte alla “durezza” dell’intera licenza (o si accetta in toto o niente) e al fatto che non permette nei fatti l’uso promiscuo di software libero e software proprietario.

Il problema era soprattutto evidente nel caso delle librerie base: nei termini della licenza non è possibile sviluppare software proprietario utilizzando librerie libere. La FSF si accorse che questo era più un limite allo sviluppo del Free Software che un’ulteriore garanzia. Lo stesso Stallman riconobbe che se il mondo commerciale avesse incominciato ad usare le librerie libere sarebbe passato più facilmente alla produzione di software libero.

1.4 La “Lesser General Purpose License”

La maggior parte del software libero, incluse molte librerie, è coperto dalla GPL, che è stata progettata principalmente per le applicazioni. Questa licenza, la “Lesser General Public License” si applica a certe specifiche librerie. Questa licenza differisce abbastanza dalla GPL, per cui è necessaria un’analisi a parte.

La ragione per cui è necessaria una licenza per le librerie è che esse vengono utilizzate in modo diverso da quello che può essere semplicemente copiare, modificare e distribuire del software. Le librerie vengono “usate” nel senso

che sono eseguite con il programma che le incorpora. Quando allego una libreria ad un programma, senza modificare la libreria, in un certo senso uso semplicemente la libreria ed è analogo ad utilizzare un qualsiasi programma. E come sappiamo l'uso del software libero è sempre garantito. Comunque, legalmente, la libreria e il codice che la usa sono un programma che estende la libreria e la GPL è nata per occuparsi di casi come questo. Ma occorre una considerazione di carattere pragmatico: a causa del fatto che la libreria viene utilizzata e distribuita nell'eseguibile, se si utilizzasse la GPL vorrebbe dire che anche il resto dell'eseguibile deve essere Free Software.

Tale clausola non agevola certo la diffusione del Free Software, perché i produttori commerciali non userebbero tali librerie.

La FSF decise di promuovere una licenza, la “Library General Purpose License” (LGPL) poi diventata “Lesser GPL”, che potesse garantire, mediante condizioni meno restrittive, l'uso delle librerie libere.

D'altronde permettere un uso indiscriminato delle librerie avrebbe privato gli utenti di quei programmi dei vantaggi del software libero. La LGPL è nata con lo scopo di permettere ai programmatori di software proprietario l'uso delle librerie e al tempo stesso di salvaguardare la libertà degli utenti di modificare le librerie incorporate in quei programmi. Tali modifiche sono da intendersi pertinenti al codice delle funzioni della libreria, non alle chiamate, o al file header, dell'eseguibile. Tale decisione fu presa con la speranza di portare a uno sviluppo più rapido delle librerie stesse.

Quello che sancisce la LGPL (in appendice con il testo originale) è la differenza di trattamento per i “lavori basati sulla libreria” e per i “lavori che usano la libreria”.

I lavori basati sulla libreria sono quei programmi che contengono codice proveniente dalla libreria, originale o modificato.

Questi programmi vengono trattati sotto le condizioni dettate dalla GPL.

I lavori che usano la libreria sono quei programmi che non contengono codice proveniente dalla libreria, originale o modificato, ma sono progettati per funzionare con la libreria mediante l'eseguibile e il linking. Tali lavori non si possono considerare derivati dalla libreria, pertanto non vengono trattati dalla LGPL. Ma il risultato finale della compilazione è un programma basato sulla libreria, perciò la licenza copre gli eseguibili di lavori che utilizzano le librerie libere.

Come vengono tutelati i diritti degli utenti in questo caso? La LGPL ga-

rantisce che l’utente deve essere messo in condizione di modificare la libreria, quindi avere copia della licenza e del sorgente, anche se ovviamente non può intervenire sul programma che “usa” la libreria.

La FSF consiglia comunque di usare il più possibile la licenza GPL soprattutto nei casi cosiddetti di importanza “strategica”. Come Stallman riporta nella sezione Phylosophy nel sito della FSF il punto di forza delle compagnie che producono software proprietario sono i soldi, il punto di forza del Free Software è la rete di programmatori che si scambiano codice. Quando fu rilasciata dalla FSF la libreria GNU Readline, che consente alla shell di trattare le stringhe di input richiamandole e modificandole, venne posta sotto i termini della GPL e non della Library GPL. La motivazione era semplice: la libreria conteneva funzionalità complesse, uniche e conferenti agli utilizzatori un vantaggio, per così dire, “strategico”. Perché allora permettere l’utilizzo di tale patrimonio a chi non produce free software? La libreria posta sotto la tutela della GPL poteva essere utilizzata solo a condizione che anche tutto il resto del programma rispettasse la GPL.

Il vantaggio del metodo di produzione Free Software viene usato per la causa stessa del Free Software!

1.5 La Open Source Initiative

La causa del Free Software sostenuta da Richard Stallman e dalla Free Software Foundation non ha incontrato i favori della maggior parte (quasi tutte) delle compagnie produttrici di software proprietario.

Il motivo è da ricercarsi nella matrice ideologica del movimento. La FSF patrocina il Free Software per sancire la prevalenza del diritto della libertà di utilizzare, modificare, distribuire ciò che è un prodotto dell’ingegno, sul diritto di proprietà dell’autore del medesimo. Tale posizione è stata tacciata, siamo in America, di “comunismo”, “anarchia” e anche di istigazione alla “pirateria” dai sostenitori del diritto di proprietà sul software.

Il caso Linux ha comunque dimostrato al mondo la validità eccezionale del metodo di produzione a sorgenti liberi, ed è tale metodo che interessa ai produttori di software.

Quando la Netscape, nel 1997, dovette trovare una strategia per non essere scalzata dal mercato da Microsoft (la famosa battaglia tra Netscape Navigator

e Internet Explorer), decise di avvalersi della collaborazione volontaria della comunità di programmatori che tanto bene aveva lavorato su Linux.

Netscape varò il progetto “Mozilla”: liberare il sorgente di Navigator.

Non si trattava di una scelta facile: come accedere al circuito del Free Software mantenendo un marchio e soprattutto mantenendo la possibilità di sfruttare economicamente il frutto del lavoro? Inoltre era necessario tutelare gli interessi delle società che avevano contribuito al Navigator fornendo codice sotto copyright.

L’uso di una licenza come la GPL avrebbe “regalato” al mondo la conoscenza sulla produzione di un buon browser ma non avrebbe giovato alle casse della Netscape, inoltre la GPL è virale: applicata a una parte del programma facilmente si estende a tutto il resto.

Occorreva un nuovo tipo di licenza. La Netscape chiese la collaborazione di Eric Raymond, membro stimato della comunità hacker e primo studioso del fenomeno Linux. Parteciparono ai lavori di creazione di una nuova licenza anche Tim O’Really, della O’Really & Associates, e Linus Torvalds, padre di Linux.

Venne prima creata la “Netscape Public License” (NPL), che incontrò molte critiche tra gli hacker ai quali si rivolgeva. In seguito a tali critiche venne rilasciata la “Mozilla Public License” (MPL), che agisce all’interno della NPL.

In pratica la NPL garantisce il diritto di sfruttamento alla Netscape sul codice originale e derivato di Navigator, la MPL garantisce come Free Software il codice nuovo.

Come nacque la “Open Source Initiative”? Eric Raymond si rese conto che l’ostacolo principale allo sviluppo di software libero erano i preconcetti tipici della lotta per fazioni. Da una parte i produttori tradizionali decisi a difendere le proprie ragioni di profitto e, al tempo stesso, desiderosi di avvalersi del patrimonio di innovazioni proprio del software libero. Dall’altra l’intransigenza di Stallman, assurto al ruolo di paladino del libero sviluppo.

Il caso Netscape ne era l’esempio.

Raymond concepì un piano di marketing per convincere i responsabili delle compagnie di software proprietario ad usufruire del software libero mettendo a loro volta a disposizione il proprio software.

Si mosse contattando Bruce Perens della Debian, una società che forniva un ambiente GNU/Linux e che aveva sviluppato il “Contratto Sociale Debian”

e la “Guida Debian al Free Software”, documenti accolti con favore dalla comunità hacker e che permettevano una possibilità commerciale. Venne redatta la “Open Source Definition” direttamente permutata da queste.

Raymond decise di usare il termine “Open Source”, per evitare l’ambiguità inglese del termine free tanto antipatica ai “proprietaristi”. Poiché non fu possibile registrare il marchio “Open Source” (troppo descrittivo) venne fondata la “Open Source Initiative” (OSI) con lo scopo di gestire la certificazione “OSI Certified”.

Quali licenze possono essere certificate come “OSI Certified”? Solo le licenze che rispondono ai canoni della “Open Source Definition”.

1.6 La “Open Source Definition”

Riporto qui di seguito il testo (in italiano) [5] con i commenti (in corsivo) tratti dai commenti originali dello stesso Bruce Perens. Il documento è tratto da “Open Source - Voci dalla rivoluzione Open Source” Apogeo, 1999.

In appendice vi è il testo originale legalmente valido.

1.6.1 Open Source Definition

Open Source non significa solo accesso al codice sorgente. I termini di distribuzione di un programma Open Source devono essere consoni ai criteri seguenti:

Si noti che la Open Source Definition non è propriamente una licenza software. È una specifica di quanto è ammesso in una licenza software perché vi si possa riferire come a un’Open Source. La Open Source Definition non è intesa per essere un documento di valore legale. L’inclusione della Open Source Definition nelle licenze software, quale quella proposta per il Progetto di Documentazione di Linux, sembra suggerire la stesura di una versione più rigorosa che sia appropriata per quell’uso. Ai fini dell’Open Source, devono applicarsi insieme tutti i termini che seguono, in tutti i casi. Per esempio, devono applicarsi alle versioni derivate di un programma così come al programma originale. Non è sufficiente applicarne alcune e non altre, e non è sufficiente se i termini non vengono applicati sistematicamente.

1. **Ridistribuzione libera** La licenza non può impedire ad alcuna parte in

causa la vendita o la cessione del software come componente di una distribuzione di software aggregato che contenga programmi proveniente da sorgenti diverse. La licenza non può richiedere diritti o il pagamento di altre concessioni per tale vendita.

Questo significa che potete fare tutte le copie che volete del software e venderle o cederle, e non dovete pagare nessuno per questo privilegio.

2. **Codice sorgente** Il programma deve includere il codice sorgente e deve consentire la distribuzione tanto in codice sorgente che in forma compilata. Laddove una qualunque forma del prodotto non sia distribuita corredata del codice sorgente, devono essere disponibili mezzi ben pubblicizzati per scaricare il codice sorgente, senza costi addizionali, via Internet. Il codice sorgente deve essere la forma preferenziale nella quale un programmatore modifichi un programma. Codice deliberatamente offuscato non è ammesso. Forme intermedie quali l'output di un preprocessore o di un traduttore non sono ammesse.

   Il codice sorgente è un preliminare necessario alla riparazione o alla modifica di un programma. L'intento qui è che il codice sorgente sia distribuito con l'opera iniziale e con tutte le opere derivate.

3. **Opere derivate** La licenza deve permettere modifiche e opere derivate e deve consentire la loro distribuzione sotto i medesimi termini della licenza del software originale.

   Il software serve a poco se non se ne può fare la manutenzione (riparazione dei bug, porting su nuovi sistemi, migliorie) e la modifica è indispensabile alla manutenzione. L'intento è qui di permettere modifiche d'ogni sorta. Deve essere permessa la distribuzione di un'opera modificata sotto gli stessi termini di licenza dell'opera originale. Tuttavia, non è richiesto che ogni produttore di un'opera derivata debba usare gli stessi termini di licenza, ma solo che possa farlo qualora lo voglia. Diverse licenze si esprimono diversamente in materia: la licenza BSD vi permette di mantenere private le modifiche, la GPL no.

   Alcuni autori di software ritengono che questa clausola possa consentire a persone prive di scrupoli di modificare il loro software in maniera che possa causare imbarazzo all'autore originale. Quello che temono è che qualcuno possa deliberatamente provocare un malfunzionamento

del software in modo che l’autore originale appaia un programmatore scadente. Altri paventano un possibile uso criminale del software tramite l’aggiunta di funzioni-cavallo di Troia o di tecnologie illegali in alcuni Paesi, come la crittografia. Tutti questi atti, tuttavia, sono coperti dal codice penale. Un comune fraintendimento a proposito delle licenze è che esse debbano specificare ogni cosa, per esempio “questo software non va usato per compiere delitti”. Dovrebbe tuttavia essere chiaro che nessuna licenza ha esistenza valida al di fuori del corpo del diritto civile e penale. Considerare una licenza come qualcosa separato dal corpo delle leggi applicabili è tanto sciocco quanto considerare un documento in lingua inglese separato dal vocabolario di quella lingua, un caso in cui nessuna parola avrebbe un significato definito.

4. **Integrità del codice sorgente dell’autore** La licenza può proibire che il codice sorgente venga distribuito in forma modificata solo se la licenza permette la distribuzione di “patch file” con il codice sorgente allo scopo di modificare il programma al momento della costruzione.

   Alcuni autori temevano che altri potessero distribuire codice sorgente con modifiche che sarebbero state percepite come opera dell’autore originale e quindi avrebbero potuto gettare ombra su di lui. Questa clausola dà loro un modo di imporre una separazione fra le modifiche e la loro opera, senza proibire le prime. C’è chi considera antiestetico che le modifiche debbano venir distribuite in un file “patch” separato dal codice sorgente, anche se distribuzioni Linux come Debian e Red Hat usano questa procedura per tutte le modifiche apportate ai programmi che distribuiscono. Esistono programmi per riversare automaticamente le patch nel sorgente principale, e questi programmi si possono eseguire automaticamente quando si scompatta un pacchetto di sorgente. Questa clausola, dunque, dovrebbe causare poca o nessuna difficoltà.

   Si noti anche che questa clausola dice che, nel caso di file patch, la modifica avviene quando si fa il build del programma. Questa scappatoia è impiegata nella Licenza Pubblica di Qt per prescrivere una diversa, anche se meno restrittiva, licenza per i file patch, in contraddizione con la sezione 3 della Open Source Definition. C’è una proposta per chiudere questa scappatoia nella definizione e mantenere nello stesso tempo Qt entro i confini dell’Open Source.

La licenza deve permettere esplicitamente la distribuzione di software costruito da codice sorgente modificato. La licenza può richiedere che le opere derivate vadano sotto nome o numero di versione differenti da quelli del software originale.

Questo significa che Netscape, per esempio, può insistere per poter essa sola chiamare una versione del programma Netscape Navigator (tm), mentre tutte le versioni gratuite del programma debbano chiamarsi Mozilla o in altro modo.

5. **Nessuna discriminazione contro persone o gruppi**

   La licenza non deve discriminare alcuna persona o gruppo di persone.

   Una licenza fornita dai Rettori dell'Università della California a Berkeley proibiva l'uso di un programma di progettazione elettronica da parte delle forze di polizia del Sud Africa. Apprezzato come merita questo sentimento in tempi di apartheid, va detto che esso non ha più senso oggi. Alcune persone si trovano ancora con software acquistato sotto quella licenza, e le loro versioni derivate devono portare la stessa restrizione. Le licenze Open Source non devono contenere tale clausola, indipendentemente dalla nobiltà dell'intento.

6. **Nessuna discriminazione di settori**

   La licenza non deve proibire ad alcuno l'uso del programma in uno specifico campo o per un determinato proposito. Per esempio, non può impedire che il programma venga usato a scopi commerciali o nella ricerca genetica.

   Il software dev'essere impiegabile allo stesso modo in una clinica che pratichi aborti e in un'organizzazione antiabortista. Queste discussioni politiche sono di pertinenza degli organi governativi, non delle licenze del software. Alcuni trovano questa mancanza di discernimento gravemente offensiva!

7. **Distribuzione della licenza**

   I diritti relativi al programma devono applicarsi a tutti coloro ai quali il programma sia ridistribuito, senza necessità di esecuzione di una licenza aggiuntiva da parte di questi.

La licenza dev’essere automatica, senza la richiesta di alcuna firma. Purtroppo, negli Stati Uniti non ci sono dati validi precedenti giudiziari del potere della licenza senza firma quando questa venga passata da una seconda a una terza parte. Tuttavia, questo argomento considera la licenza come facente parte della legge sul contratto, mentre qualcuno obietta che dovrebbe essere considerata come legge di copyright, campo in cui si danno più precedenti per quel tipo di licenza. Un buon precedente ci sarà senz’altro nei prossimi anni, data la popolarità del questa licenza e il boom dell’Open Source.

8. **La licenza non dev’essere specifica a un prodotto**

   I diritti relativi a un programma non devono dipendere dall’essere il programma parte di una particolare distribuzione software. Se il programma è estratto da quella distribuzione e usato o distribuito entro i termini della licenza del programma stesso, tutte le parti a cui il programma sia ridistribuito dovrebbero avere gli stessi diritti che vengono garantiti in unione alla distribuzione software originale.

   Questo significa che non si può impedire a un prodotto identificato come Open Source di essere gratuito solo se lo si usa con una marca particolare di distribuzione Linux, ecc. Deve rimanere gratuito se anche lo si separa dalla distribuzione software da cui proviene.

9. **La licenza non deve contaminare altro software**

   La licenza non deve porre restrizioni ad altro software che sia distribuito insieme a quello licenziato. Per esempio, la licenza non deve pretendere che tutti gli altri programmi distribuiti sullo stesso media siano software Open Source.

   Una versione di GhostScript (programma di rendering PostScript) richiede che i media sui quali viene distribuito contengano solo programmi software gratuiti. Questo non è consentito dalla licenza Open Source. Per fortuna, l’autore di GhostScript distribuisce un’altra versione del programma (un po’ più vecchia) sotto una licenza Open Source genuina.

   Si noti che c’è differenza fra derivazione e aggregazione. Derivazione è quando un programma incorpora di fatto in sé parti di un altro programma. Aggregazione è quando due programmi vengono inclusi

sullo stesso CD-ROM. Questa sezione della Open Source Definition riguarda l'aggregazione, non la derivazione. La sezione 4 riguarda la derivazione.

10. **Licenze esemplari**

    Le licenze GNU GPL, BSD, X Consortium e Artistica sono esempi di licenze da considerarsi conformi alla Open Source Definition. Altrettanto dicasi della MPL.

    Questo sarebbe una fonte di guai nel giorno in cui una di queste licenze si modificasse e non fosse più Open Source: si dovrebbe pubblicare immediatamente una revisione della Open Source Definition. Ciò è pertinente per la verità al testo esplicativo, non alla Open Source Definition in sé.

1.7 Il primo risultato dell'OSI

Il gruppo KDE e Troll Tech cercarono di porre un prodotto non-Open Source entro l'infrastruttura di Linux, incontrando una resistenza inattesa. Le grida di pubblico scandalo e la minaccia che il loro prodotto venisse rimpiazzato da un altro, completamente Open Source, convinse alla fine Troll Tech a convertirsi a una licenza pienamente Open Source. È un segno interessante dell'accoglienza entusiastica riservata dalla comunità alla Open Source Definition il fatto che Troll dovette adeguare la propria licenza, pena l'insuccesso del suo prodotto.

KDE fu il primo esperimento di un desktop grafico gratuito per Linux. Le applicazioni KDE erano esse stesse sotto GPL, ma dipendevano da una libreria grafica proprietaria nota come Qt, di Troll Tech. I termini della licenza di Qt ne proibivano la modifica o l'uso con qualunque display software che non fosse il senescente X Window System. Ogni uso diverso richiedeva allo sviluppatore una licenza del costo di 1500 dollari. Troll Tech fornì versioni di Qt per Windows di Microsoft e per Macintosh, e questa fu la sua principale fonte d'entrate. La licenza pseudo-gratuita per i sistemi X intendeva indirizzare i contributi degli sviluppatori Linux verso demo, esempi e accessori per i loro costosi prodotti Windows e Mac.

Per quanto i problemi della licenza di Qt apparissero evidenti, la prospettiva di un desktop grafico per Linux era così attraente che molti utenti furono

disposti a chiudere un occhio sulla sua natura non-Open Source. I promotori di Open Source trovarono che KDE fosse in difetto perché avevano l'impressione che gli sviluppatori stessero cercando di confondere la definizione di free software allo scopo di includervi elementi solo parzialmente gratuiti, come Qt. Gli sviluppatori KDE replicarono che i loro programmi erano Open Source, anche se non esistevano versioni eseguibili di quei programmi che non richiedessero una libreria non-Open Source. Bruce Perens e altri sostennero che le applicazioni KDE non erano che frammenti Open Source di programmi non-Open Source, e che una versione Open Source di Qt sarebbe stata necessaria prima che ci si potesse riferire a KDE come a un Open Source.

Gli sviluppatori KDE tentarono di risolvere parzialmente il problema della licenza di Qt negoziando con Troll Tech un accordo (KDE Free Qt Foundation) in cui Troll e KDE avrebbero congiuntamente controllato i rilasci delle versioni gratuite di Qt, e Troll Tech avrebbe rilasciato Qt sotto una licenza conforme a Open Source nel caso che l'azienda venisse acquisita o cessasse l'attività.

Un altro gruppo diede inizio al progetto GNOME, un concorrente interamente Open Source di KDE che mirava a fornire maggiori funzioni e sofisticazioni; un gruppo separato avviò il progetto Harmony per produrre un clone di Qt completamente Open Source che avrebbe supportato KDE. Mentre le dimostrazioni di GNOME avvenivano fra il plauso e Harmony stava per diventare usabile, Troll Tech capì che QT non avrebbe riscosso successo nel mondo Linux se non avesse cambiato licenza. Troll Tech rilasciò dunque una licenza interamente Open Source per Qt, disinnescando il conflitto ed eliminando i motivi alla base del progetto Harmony. Il progetto GNOME continua tuttora, volto adesso a un KDE migliore in termini di funzionalità e di raffinatezza piuttosto che in termini di licenza.

Quando il sistema GNOME sarà completo, sarà stato realizzato un sistema operativo con desktop GUI Open Source in grado di competere con Microsoft NT.

1.8 Rapporti tra FSF e OSI

Con la nascita della Open Source Initiative si è avuto un dibattito, con toni a volte aspri, tra i leader dei due movimenti: Eric Raymond per la OSI e Richard Stallman per la FSF. Anche la comunità hacker ha rischiato una pericolosa

scissione fino a quando non è stato chiaro l'intento dell'OSI: promuovere il Free Software e non svenderlo agli interessi di mercato.

Tale dibattito ha avuto luogo in conferenze e sui newsgroup dedicati. Fu pacificamente sospeso dai protagonisti stessi quando si accorsero che rischiava di diventare una diatriba senza soluzione. Quale poteva essere il motivo del disaccordo? Non solo uno purtroppo. Come già detto per Stallman il discorso sul Free Software è un discorso sulla libertà dell'individuo, sulla responsabilità, sullo spirito di collaborazione. Tutto il resto è secondario. Stallman sostiene che il lavoro di creazione del software deve essere retribuito nella misura in cui è utile alla società, e quindi, per essere utile, deve essere liberamente fruibile da tutti. Il software viene visto come un'opera della ricerca scientifica: non deve essere brevettabile come non è brevettabile una formula matematica, deve essere distribuito allo stesso modo delle ipotesi, dei metodi, dei risultati di un esperimento scientifico poiché questo è l'unico modo di verificare se l'esperimento prova la tesi. Le ipotesi, i metodi e i risultati sono nel software il codice sorgente. Ecco quindi la necessità di distribuire il sorgente e non, o non solo, l'oscuro eseguibile. Inoltre, secondo Stallman, lavorare per sviluppare software proprietario è un grave delitto contro la solidarietà sociale. Infatti essendo il programmatore vincolato al segreto industriale non può aiutare altri programmatori magari alle prese con gli stessi problemi. Non solo, questa situazione porta ad un grave spreco di risorse: quante volte in informatica sono stati reinventati il fuoco e la ruota? Ossia: quante volte si sono risolti problemi già risolti altrove? Almeno una volta per compagnia. Non ultimo Stallman ritiene il software proprietario come il nemico da battere.

Le opinioni di Stallman sono condivisibili ma sollevano molte obiezioni, ad esempio: come faranno a vivere i programmatori se non vengono stipendiati? Stallman risponde che a) Se si proibissero i proventi derivanti da ogni forma di occupazione musicale, la gente smetterebbe di cantare? Probabilmente no, così neanche si smetterebbe di programmare. b) Software libero non significa software gratis. Si potrebbe richiedere un compenso per la distribuzione del software, per le attività di consulenza, di assistenza, di manutenzione, ecc.

Raymond parte da un discorso più pragmatico: lo sviluppo a sorgenti liberi ha dimostrato (vedere il capitolo 3) di essere un potente mezzo di sviluppo, forse oggi il più potente. Perché non consentire di accedere a questo mezzo anche a chi sviluppa software proprietario considerando che, per accedervi,

dovrebbe a sua volta fornire software libero? E poi: considerato che chi fa mercato non è culturalmente pronto ad accettare i principi ma vuole adottare i metodi che da essi derivano, perché impedirlo?

Le due posizioni si confrontarono su una questione di principio, su una questione personale e su una questione di prospettive. Mi limiterò a brevi cenni sulle tre, non ritenendo utile allo scopo della tesi una dissertazione più approfondita.

Stallman rimproverò a Raymond di aver rinunciato al concetto di libertà in favore del concetto di mercato. Raymond rispose che lo scopo dell'OSI è diffondere il software libero, non partecipare alle crociate. La conversione di una mentalità di mercato passa per i mezzi che questa adotta per vivere nel mercato stesso. Dimostrando che il software libero può essere economicamente conveniente si dimostra anche la validità dei principi da cui nasce.

Stallman si ritenne "cassato dalla storia", perché benché fosse stato lui il primo promotore del Free Software si vide relegato a ruoli marginali nel mondo del Free Software. Probabilmente accadde che nell'opera di marketing concepita da Raymond la presenza di una figura come Stallman avrebbe allontanato proprio chi si cercava di avvicinare. Questo potrebbe spiegare come mai Raymond si mosse, come dire, "autonomamente".

Stallman non credeva nella creazione di un marchio di certificazione "Open Source" ritenendo che avrebbe potuto essere facilmente "forzato" o sfruttato dalle compagnie commerciali. E questo è quello che è avvenuto. Alcune società, ottenuto il marchio su certi prodotti, hanno provveduto ad allargarlo a prodotti non Free Software. Il tutto con rispetto delle norme vigenti il marchio ma cercando di imporre prodotti non free confondendo l'utente sprovveduto.

La discussione appartiene ormai al passato. Come convivono oggi le due organizzazioni?

È lo stesso Stallman a tracciare il quadro della situazione [6]:

> "Il movimento Free Software e il movimento Open Source sono come due partiti politici entro la comunità hacker.
>
> I gruppi radicali sono conosciuti per la faziosità: le organizzazioni si scindono a causa di opinioni diverse sui dettagli della strategia da seguire, e si odiano l'un l'altra. Sono d'accordo sui principi base ma non sulla loro applicazione; si considerano nemiche l'una dell'altra e si combattono senza quartiere.

> Per il movimento Free Software e il movimento Open Source è esattamente il contrario. Siamo in disaccordo sui principi base ma convergiamo nella pratica. Lavoriamo insieme su molti progetti.
>
> Noi del movimento Free Software non consideriamo il movimento Open Source come un nemico. Il nemico è il software proprietario. Ma allo stesso tempo ci teniamo a far sapere che non siamo uguali!”

Pace è fatta.

2 Nel regno degli Hacker

2.1 Introduzione

Questo capitolo intende narrare la genesi dell'idea dell'Open Source e della nascita dell'organismo che la tutela: la "Open Source Initiitive". L'idea di Open Source, e soprattutto di Free Software, viene vissuta nei laboratori del pionerismo informatico senza averne una reale consapevolezza. Lo scambiarsi codice, il correggerlo, estenderlo, sono gli strumenti del rapido sviluppo di questa nuova branca della scienza, non una precisa scelta morale. Il concetto di proprietà del software era sentito come un riconoscimento di paternità, non uno strumento di profitto. Esattamente come un teorema per un matematico. Quando il software uscì dai laboratori per essere impiegato a fini produttivi molti ricercatori vennero assorbiti dalla nascente industria e si perse il concetto di software come bene scientifico. Ma non tutti desistettero. Negli anni '80, utilizzando le risorse del laboratorio di intelligenza artificiale del MIT, un ricercatore, Richard Stallman, non cedette alle lusinghe economiche del software commerciale e iniziò la sua personale battaglia per sancire il diritto al software liberamente disponibile. Non resta solo per molto, altri programmatori aderiscono al progetto e incominciano ad uscire i primi prodotti. Con i primi risultati altri programmatori si aggregano via Internet, sono gli hacker, cioè dilettanti, professionisti, ricercatori, autodidatti che hanno un'eccellente conoscenza dei sistemi informatici che utilizzano. Il progetto di Stallman, "GNU", decolla e oggi può distribuire a chiunque lo richiede un ambiente completamente compatibile con Unix.

Negli anni '90 uno studente finlandese, Linus Torvalds, lancia e gestisce la costruzione di un kernel UNIX-like, "Linux", il progetto cresce aldilà di ogni previsione alimentato dai contributi, sotto forma di codice e testing, degli hacker.

Dal 1995 GNU e Linux vengono distribuiti insieme fornendo un intero ambiente operativo, gratuito, Free Software. L'idea di Free Software esce co-

sì dai meandri del Web e viene spinta avanti dalla forza dei risultati: rapido tasso di sviluppo, innovazione, stabilità, costi molto bassi. Il modello attira investitori che potrebbero fornire mezzi per affrontare altri progetti ma la fermezza di Stallman nel difendere l'idea scoraggia ogni approccio. È tempo, siamo nel 1997, di un nuovo passo in avanti: la "Open Source Initiative".

La Open Source Initiative nasce con lo scopo di permettere l'avvicinamento di due culture e la fusione di due patrimoni. La cultura del software proprietario, figlia di società grandi e piccole che hanno fatto del software una fonte di reddito, e la cultura del Free Software, nata in storici laboratori di ricerca e auto gestita da fantomatici personaggi: gli hacker.

2.1.1 La tribù Hacker

L'hacker è l'esperto del sistema. Colui che conosce nei dettagli l'architettura di un sistema informatico al punto da poter intervenire mutandola per i propri scopi. È un esperto di programmazione. Conosce i kernel, i protocolli di comunicazione e gli strumenti che gli permettono di intervenire su questi.

Per essere Hacker non importano i titoli di studio o le qualifiche professionali. Non esiste un patentino. Si è hacker quando altri hacker Ti chiamano Hacker. È una tribù governata con criterio meritocratico elevato all'ennesima potenza.

L'accesso alla comunità è molto semplice: un qualsiasi newsgroup di programmatori riguardante Linux (per esempio) è già territorio hacker ("Hackerdom"). La partecipazione è immediata, scorrono mail riguardanti problemi tecnici e soluzioni, domande, risposte e commenti. È possibile intervenire in qualsiasi momento, ma attenzione! Se si è nuovi non vi è possibilità di errore. Permettetevi un solo errore e verrete derisi e insultati (sono un po' cattivelli) in ogni angolo del web. Le castronerie adorneranno gli archivi di barzellette e saranno riportate ad esempio della disumana inettitudine di chi ha provato senza essere all'altezza. Trattamento diverso per i commenti sensati: si viene interrogati in tono inquisitorio, l'osservazione viene analizzata parola per parola, seguono richieste di chiarimenti atte a verificare le reali capacità dell'autore. Se si passa l'esame allora si è sulla buona strada. Il vero banco di prova è il codice: se si vuole essere Hacker bisogna produrre codice perfetto. È possibile partecipare a un qualsiasi progetto scaricando i sorgenti e incominciando a studiarli. Se si trovano errori e si è in grado di correggerli

allora conviene mandare il proprio codice al gruppo che gestisce il progetto che sottoporrà a una prima revisione il programma, passata l'analisi iniziale il codice viene allegato alla distribuzione principale, da questo momento sarà sottoposto all'analisi di centinaia di occhi esperti. Stessa strada seguono le modifiche perfettive o l'implementazione di nuove funzionalità.

Il codice, sotto l'esame della comunità, viene discusso e ulteriormente migliorato. Quando un contributo viene accettato allora si ha l'onore di poter inserire il proprio nome nella lista dei collaboratori al progetto (ogni progetto ne ha una, è un dato di fatto).

Chi porta più contributi o i contributi migliori acquista considerazione da parte degli altri programmatori fino a diventare una figura di riferimento. La gestione dei progetti, cioè il mantenimento del codice sorgente, dei newsgroup, delle mailing list relative, è affidato a chi dimostra di conoscere meglio l'intero progetto.

Il gradino ultimo dell'ascesa sociale nell'hackerdom è diventare responsabile di un progetto. Più il progetto è complesso e importante maggiore è il prestigio per chi lo gestisce. Il prestigio e il riconoscimento all'interno della comunità sono l'unica ricompensa, così come la derisione e l'insulto sono l'unica punizione.

È un tipo di società in cui i cittadini sono al tempo stesso operai, giudici e poliziotti. Tutti producono codice, tutti giudicano codice, tutti controllano che commenti e contributi siano all'altezza della situazione.

Cosa tiene legati gruppi eterogenei e senza praticamente alcun contatto che non sia la posta elettronica?

Perché accade solo in casi eccezionali che due gruppi portino avanti lo stesso progetto in concorrenza?

Raymond ha analizzato il problema in "Colonizzare la Noosfera" [7]:

> La "noosfera" indicata nel titolo di questo saggio è il territorio delle idee, lo spazio dove convivono tutte le concezioni possibili.

È in questo spazio che vivono gli hacker. Un progetto parte da una necessità che diventa idea su come porvi rimedio. Il progetto di lavoro è l'applicazione dell'idea. Come è possibile organizzarsi all'interno di un habitat come Internet?

2.2 La cultura del dono

Invito innanzitutto a considerare che la programmazione per gli hacker è in primo luogo gratificazione dell'ego e solo a volte diventa, in aggiunta, leva di reddito. Allo stesso modo del canto o della pittura per gli artisti.

Gli hacker come società hanno dovuto affrontare i problemi tipici delle società composte da individui indipendenti: la gestione del territorio e la distribuzione delle risorse.

In Internet il territorio è pressoché infinito e soprattutto impossibile da delimitare. Allo stesso modo risorse quali potenza di calcolo, memoria, banda passante non costituiscono oggi un limite.

In un ambiente in cui c'è abbondanza di tutto e non esistono reali pericoli di sopravvivenza cosa viene ritenuto elemento di distinzione sociale, ossia da cosa viene sancita la gratificazione dell'Ego? Non dalla ricchezza, perché tutti hanno già quello che vogliono. Non dal potere coercitivo, impossibile da esercitarsi su Internet.

In sociologia si ritiene tipico delle società abitanti territori ricchi di risorse organizzarsi secondo un modello basato sulla "cultura del dono". In tali società la scala di reputazione sociale è basata sulla capacità di donare degli individui.

In genere viene individuata una risorsa di aggregazione, come una fonte di acqua potabile, che viene gestita dall'individuo in qualche modo meritevole, per esempio dallo stesso individuo che l'ha scoperta o dal più anziano.

Il gestore della risorsa si adopera perché essa continui a rimanere fruibile dalla comunità, ne fa "dono" alla comunità. In cambio riceve stima e considerazione, come "dono", da parte degli altri individui. Quindi non c'è un reale scambio di beni ma un mutuo riconoscimento di stima e considerazione. Non essendoci altra moneta questo diviene la ricompensa, la gratificazione dell'ego.

Tra gli hacker abbiamo un processo analogo: il progetto è la sorgente alla quale gli individui fanno riferimento come punto di aggregazione per ricevere e dare stima. Chi gestisce il progetto è l'individuo che più ha possibilità di dare e di ricevere stima, essendo lui quello che regola l'accesso alla fonte. Tra tanti progetti tra cui scegliere gli individui cercheranno il progetto che lascia loro più spazio da colonizzare, per avere più possibilità di diventare a loro volta gestori di fonte, e che è gestito da un "giusto" gestore, per essere sicuri

di ricevere i meriti spettati. Questo porta i gestori a essere imparziali e corretti nelle loro decisioni.

Ogni nuovo progetto è un campo di idee virtualmente inesplorato, i primi ad aderire avranno la possibilità di guadagnare più spazi di competenza (possibili sorgenti) e quindi di ricevere gratificazione. È anche vero che un progetto appena abbozzato è un progetto ad altissimo rischio di fallimento ed è raro che qualcuno vi collabori finché non è stato realizzato almeno un nucleo stabile. La noosfera è allettante agli hacker nella misura in cui prevedono di ricavarne affermazioni. Progetti difficili, molto conosciuti e già avviati danno garanzia di "buon guadagno".

In questo tipo di economia sono rari gli scontri che non giungano ad una pacifica conclusione. Il conflitto è, in generale, la risoluzione di un processo di spartizione. Gli hacker si spartiscono il pubblico riconoscimento per aver collaborato a un progetto. Tale riconoscimento è proporzionale alle dimensioni e alla difficoltà del progetto. Il progetto è il patrimonio da cui attingere e la sua crescita diventa la principale attenzione dei collaboratori, necessità superiore agli interessi personali.

È possibile che avvengano scontri di giudizio all'interno della comunità su quali scelte adottare. Tali questioni vengono risolte procedendo per competenza e per anzianità di lavoro, in caso di parità decide il gestore del progetto.

È anche raro che vengano sviluppati progetti analoghi da gruppi diversi, la concorrenza non è vista di buon occhio perché limita la possibilità di acquisire riconoscimenti e inoltre divide le forze. Agli hacker non interessa imporre un prodotto, interessa guadagnare prestigio all'interno della comunità. Per ottenere questo è necessario che più gente possibile sappia su cosa si lavora, se due gruppi portano avanti in parallelo lo stesso progetto il pubblico è dimezzato.

Particolare attenzione viene posta quando il gestore di un progetto trasferisce il controllo sul progetto a un altro hacker. L'evento viene largamente pubblicizzato con mesi di anticipo per garantire che tutti gli interessati sappiano e al tempo stesso per legittimare il successore presso i collaboratori.

Altra situazione quando il gestore del progetto abbandona il progetto senza passare il testimone. In genere succede che, dopo essere sicuro che il progetto è veramente senza guida, qualcuno si faccia avanti come volontario. È un'operazione delicata che richiede molte attenzioni. Non basta auto legit-

timarsi come guida di un progetto, è necessario che i collaboratori siano d'accordo. Così in genere chi desidera acquisire un progetto orfano deve, dopo averne annunciato l'intenzione più volte, aspettare di raccogliere un generale consenso.

Gli hacker oggi vivono in Internet, ma non è sempre stato così .

Eric S. Raymond [8] illustra le origini della tribù di programmatori da cui scaturisce prima come sentimento, poi come costruzione legale, il concetto di Free Software e di Open Source. Essendo questa una delle poche testimonianze dirette sull'alba dell'informatica e sugli sviluppi conseguenti ne riporterò un ampio stralcio.

2.3 Le origini

Fin dal 1945 la tecnologia informatica ha attirato molte delle menti più brillanti e creative del pianeta. A partire dall'ENIAC di Eckert e Mauchly in avanti, è sempre esistita cultura tecnica più o meno continua e autocosciente di programmatori entusiasti, di persone il cui rapporto col software era di puro divertimento.

I primi programmatori di solito provenivano dai settori dell'ingegneria e della fisica. Sono dipinti come tipi un po' particolari: calzini bianchi, camicie e cravatte in poliestere e lenti spesse. Nulla di inusuale per chi è abituato al Poli!

Programmavano in linguaggio macchina, in FORTRAN e in un'altra mezza dozzina di linguaggi ormai dimenticati. Si tratta dei precursori della cultura hacker, dei ben poco celebrati protagonisti della sua preistoria. Questi primi programmatori furono nominati, negli anni '80, i "Real Programmer"

Dalla fine della Seconda Guerra Mondiale ai primi anni '70, nella grande era dei computer a linea di comando e dei mainframe chiamati "big iron", i Real Programmer dominarono la scena della cultura tecnica dei computer.

Ciò che ebbe origine dalla cultura "Real Programmer", è lo slancio innovativo che investì il computer interattivo, le università e le reti. Elementi che hanno avuto un ruolo fondamentale nella nascita di una tradizione tecnica che sarebbe sfociata nell'attuale cultura hacker Open Source.

2.4 I primi hacker

L'origine della cultura hacker, come oggi la conosciamo, può essere fatta risalire al 1961, anno in cui il MIT acquistò il primo PDP-1. Il comitato Signals and Power del Club Tech Model Railroad del MIT, adottò la macchina quale prediletto giocattolo-tecnologico creando strumenti di programmazione, linguaggi e quell'intera cultura che ancora oggi ci appartiene in modo inequivocabile. Questi primi anni sono stati esaminati nella prima parte del libro Hackers di Steven Levy (Anchor/Doubleday, 1984).

La cultura informatica del MIT sembra essere stata la prima ad adottare il "termine hacker". Gli hacker della TMRC divennero il nucleo dell'Artificial Intelligenge Laboratory (Laboratorio di Intelligenza Artificiale) del MIT, il principale centro di ricerca AI (Intelligenza Artificiale) su scala mondiale, nei primi anni '80. La loro influenza si protrasse ben oltre il 1969, il primo anno di ARPAnet.

ARPAnet è stata la prima rete transcontinentale di computer ad alta velocità. Ideata e realizzata dal Ministero della Difesa statunitense come esperimento nelle comunicazioni digitali, crebbe fino a diventare un collegamento tra centinaia di università, esponenti della difesa e laboratori di ricerca. Permise a tutti i ricercatori, ovunque essi si trovassero, di scambiarsi informazioni con velocità e flessibilità senza precedenti, dando un forte impulso allo sviluppo del lavoro di collaborazione e accelerando enormemente il ritmo e l'intensità del progresso tecnologico.

Ma ARPAnet fece anche qualcos'altro. Le sue autostrade elettroniche misero in contatto gli hacker di tutti gli Stati Uniti e questi, finora isolati in sparuti gruppi, ognuno con la propria effimera cultura, si riscoprirono (o reinventarono) nelle vesti di vera a propria tribù di rete.

Le prime intenzionali azioni di hackeraggio - i primi linguaggi caratteristici, le prime satire, i primi dibattiti autocoscienti sull'etica hacker - tutto questo si propagò su ARPAnet nei suoi primi anni di vita. (Basti come esempio la prima versione del Jargon File, datato 1973.) La cultura hacker mosse i primi passi nelle università connesse alla Rete, in particolar modo (ma non esclusivamente) nei loro dipartimenti di scienza informatica.

Dal punto di vista culturale, l'AI (Intelligenza Artificiale) Lab del MIT è da considerarsi il primo tra i laboratori di pari natura a partire dai tardi anni '60. Anche se istituti come il Laboratorio di Intelligenza Artificiale dell'Università di Stanford (SAIL) e, più tardi, l'Università Carnegie-Mellon (CMU),

divennero in seguito quasi altrettanto importanti. Tutti costituivano fiorenti centri di scienza dell'informazione e ricerca sull'intelligenza artificiale. Tutti attiravano individui brillanti che contribuirono al grande sviluppo del mondo degli hacker, sia dal punto di vista tecnico che folkloristico.

Per comprendere ciò che successe dopo, comunque, è necessario un ulteriore sguardo ai computer stessi, poiché sia la nascita del Laboratorio che il suo futuro declino furono fortemente influenzati dalle correnti di cambiamento nell'ambito della tecnologia informatica.

Fin dai giorni del PDP-1, le sorti dell'hacking si intrecciarono alla serie di minicomputer PDP della Digital Equipment Corporation (DEC). La DEC aprì la strada a prodotti interattivi di stampo commerciale ed a sistemi operativi time-sharing. La flessibilità, la potenza e la relativa economicità di queste macchine, portarono molte università al loro acquisto.

L'economicità dei sistemi time-sharing, costituì l'habitat ideale per lo sviluppo della cultura hacker e anche ARPAnet fu costituita, per la maggior parte della sua durata, da una rete di macchine DEC.

La più importante fra queste fu il PDP-10 che fece la sua comparsa nel 1967. Essa rappresentò la macchina preferita dagli hacker per quasi quindici anni; il TOPS-10 (sistema operativo DEC per la macchina) e il MACRO-10 (suo assemblatore), sono ancora ricordati con passione nostalgica nell'ambito della cultura hacker.

Il MIT, pur utilizzando lo stesso PDP-10, imboccò una strada lievemente diversa; rifiutò il software DEC del PDP-10 scegliendo di creare un proprio sistema operativo, il leggendario ITS.

ITS stava per "Incompatible Timesharing System", (Sistema Time-Sharing Incompatibile), sigla che rendeva perfettamente l'idea delle intenzioni insite nel progetto: volevano fare a modo loro. Fortunatamente per tutti noi, la gente della MIT possedevano un grado di intelligenza in grado di contrastare la sua arroganza. L'ITS, strambo, eccentrico e a volte perfino pieno di difetti, portò tuttavia una brillante serie di innovazioni tecniche, e ancora detiene senza dubbio il record di sistema operativo time-sharing più a lungo utilizzato.

Lo stesso ITS fu scritto in Assembler, ma molti progetti ITS furono scritti nel linguaggio LISP dell'AI. Il LISP si rivelò il più potente e flessibile linguaggio dell'epoca e, a distanza di vent'anni, si presenta ancora meglio congegnato rispetto a molti dei linguaggi odierni. Il LISP permise agli hacker di ITS di dare libero sfogo a tutta la loro creatività. Fu forse questa la formu-

la del successo straordinario di questo linguaggio, che resta uno dei preferiti dagli hacker.

Molte creazioni tecniche della cultura ITS, sopravvivono ancora oggi; l'editor Emacs è forse il più conosciuto. Così come molto del folklore ITS è tuttora "vivo" per gli hacker, come dimostra il Jargon File.

Non si può certo dire che il SAIL e il CMU si fossero nel frattempo assopiti. Molti nuclei di hacker, sviluppatisi intorno al PDP-10 del SAIL, divennero più tardi figure chiave nel progresso dei personal computer e delle interfacce di software finestra/icona/mouse, come oggi le conosciamo. Gli hacker di CMU, dal canto loro, stavano portando avanti ciò che avrebbe dato vita alle prime applicazioni pratiche su larga scala di sistemi esperti e di robotica industriale.

Un altro luogo che ha giocato un ruolo fondamentale per il progresso culturale fu lo Xerox PARC, il famoso Centro Ricerche di Palo Alto. Per più di un decennio, a partire dai primi anni '70 fino alla metà degli '80, il PARC produsse un'impressionante quantità di innovazioni hardware e software. Le moderne interfacce di software costituite da mouse, finestre e icone, videro la luce proprio in quell'ambito, ma anche le stampanti laser e la local area network (LAN). La serie PARC di macchine D, anticipò di un decennio i potenti personal computer degli anni '80. Purtroppo, questi profeti non ebbero né onori né gloria in seno alla loro azienda e presto diventò un'abitudine descrivere sarcasticamente il PARC come un luogo caratterizzato dallo sviluppo di brillanti idee per chiunque altro, tranne che per se stessi. L'influenza di queste menti sulla cultura hacker fu comunque a dir poco pervasiva.

Le culture ARPAnet e PDP-10 crebbero in forza e varietà nell'arco degli anni '70. I programmi per le mailing list elettroniche, utilizzati fino ad allora per incoraggiare la cooperazione tra i diversi gruppi di interesse disseminati a quattro angoli del mondo, furono sempre più impiegati per scopi sociali e ricreativi. DARPA chiuse deliberatamente un occhio di fronte alle attività tecniche "non-autorizzate", ben comprendendo come queste spese extra fossero un piccolo prezzo da pagare rispetto all'effetto di convogliare l'attenzione di un'intera generazione di menti giovani e brillanti alla causa dell'informatica.

Probabilmente, la più nota delle mailing list a sfondo "sociale" di ARPAnet fu la SF-LOVERS, per gli appassionati di fantascienza; basti pensare che ancora oggi essa continua ad esistere in "Internet", l'erede naturale e senza confini della rete ARPAnet. In questo scenario, si contano numerosi altri pio-

nieri di questo stile di comunicazione che più tardi venne commercializzato in servizi time-sharing a pagamento come CompuServe, Genie e Prodigy.

2.5 La nascita di Unix

Nel frattempo, comunque, nel New Jersey, qualcos'altro era stato messo in cantiere fin dal 1969, qualcosa che avrebbe inevitabilmente adombrato la tradizione del PDP-10. L'anno di nascita di ARPAnet, fu anche l'anno in cui un hacker dei Laboratori Bell, di nome Ken Thompson, inventò il sistema Unix.

Thompson si era trovato coinvolto nella fase di sviluppo di un Sistema Operativo Time-Sharing chiamato Multics, che divideva la propria discendenza con ITS. Multics costituì un importante banco di prova su come la complessità di un sistema operativo potesse essere celata fino a essere resa invisibile all'utente e perfino alla maggioranza dei programmatori. L'idea fu quella di rendere l'uso di Multics molto più semplice e programmabile in modo da permettere di operare anche dall'esterno.

I Laboratori Bell si tirarono fuori dal progetto quando Multics iniziò a mostrare segni di crescita non giustificata (il sistema fu poi commercializzato da Honeywell, senza successo). Ken Thompson cominciò ad avere nostalgia dell'ambiente Multics, e pensò di giocare un po' miscelando alcune caratteristiche del sistema operativo naufragato con altre di sua concezione su un rottame di DEC PDP-7.

Un altro hacker, di nome Dennis Ritchie, inventò un nuovo linguaggio chiamato "C", da usare con una versione Unix di Thompson ancora allo stato embrionale. Come Unix, C fu progettato per essere piacevole e facile da usare oltre che flessibile. L'interesse per questi strumenti non tardò a crescere nell'ambito dei Laboratori Bell, e subì un'impennata nel 1971 quando Thompson e Ritchie vinsero un appalto per produrre quello che oggi chiameremmo sistema di office-automation per uso interno. Ma Thompson e Ritchie avevano in mente qualcosa di ben più ambizioso.

Per tradizione, i sistemi operativi erano stati, fino ad allora, scritti in Assembler in modo da ottenere la maggiore efficienza possibile dalle macchine host. Thompson e Ritchie furono tra i primi a capire che la tecnologia dell'hardware e dei compilatori aveva raggiunto un tale livello di maturità da poter scrivere in C un intero sistema operativo: nel 1974 l'intero am-

biente operativo era regolarmente installato su numerose macchine di diversa tipologia.

Si tratta di un evento senza precedenti e le implicazioni che ne derivarono furono enormi. Se davvero Unix poteva presentare la stessa interfaccia e le stesse funzionalità su macchine di diverso tipo, era sicuramente in grado di fungere da ambiente software comune per tutte. Gli utenti non avrebbero mai più dovuto pagare per nuovi software appositamente progettati ogni volta che una macchina diventava obsoleta. Gli hacker erano in grado di utilizzare gli stessi strumenti software da una macchina all'altra, piuttosto che dover reinventare l'equivalente di fuoco e ruota ogni volta.

Oltre alla portabilità, Unix e C presentavano altri punti di forza. Entrambi si basavano sulla filosofia "Keep it simple, stupid!" letteralmente "Semplifica, stupido!". Un programmatore poteva senza difficoltà tenere a mente l'intera struttura logica di C (a differenza di molti altri linguaggi precedenti, ma anche successivi), e non dover più ricorrere continuamente ai manuali. Unix era un insieme flessibile di semplici strumenti che si mostravano complementari l'un l'altro.

Questa combinazione si rivelò adatta per una vasta gamma di operazioni, incluse alcune completamente nuove, non previste in origine dagli stessi progettisti. La sua diffusione in AT&T fu estremamente rapida, a dispetto della mancanza di programmi di supporto formale. Entro il 1980, il suo uso si era già allargato a un gran numero di università e siti di ricerca informatica, e centinaia di hacker la consideravano come la propria casa.

Le macchine da lavoro della prima cultura Unix furono i PDP-11 e il loro discendente fu il VAX. Ma, proprio per la sua caratteristica portabilità, Unix funzionava senza alcuna modifica su una vasta gamma di macchine che costituivano ARPAnet. Nessuno usava l'Assembler, i programmi creati in C erano facilmente utilizzabili su tutte queste macchine.

Unix aveva persino una propria rete non certo di qualità eccelsa: Unix-to Unix Copy Protocol (UUCP), bassa velocità, poco affidabile ma economica. Due macchine Unix qualsiasi potevano scambiarsi posta elettronica point-to-point attraverso le ordinarie linee telefoniche. Questa funzionalità era parte integrante del sistema e non solo un'opzione. Le postazioni Unix cominciarono a formare una rete a se stante, e una cultura hacker iniziò a crescere al suo interno. È del 1980 la prima Usenet board, che sarebbe rapidamente diventata più grande di ARPAnet.

ARPAnet stessa ospitò alcuni siti Unix. PDP-10 e le culture Unix e cominciarono a incontrarsi e fondersi, anche se, dapprima, senza grande successo. Gli hacker di PDP-10 consideravano la gente di Unix come una banda di principianti che utilizzava strumenti dall'aspetto primitivo, se paragonati alla squisita e perfino barocca complessità di LISP e ITS. "Coltelli di pietra e pelli d'orso!" brontolavano.

Ecco allora che si delineò un terzo scenario. Il primo personal computer fu immesso sul mercato nel 1975. La Apple fu fondata nel 1977, e il suo progresso avvenne con impressionante rapidità negli anni che seguirono. Il potenziale dei microcomputer era ormai chiaro e attrasse inevitabilmente un'altra generazione di giovani e brillanti hacker. Il loro linguaggio era il BASIC, talmente primitivo che i partigiani del PDP-10, e gli aficionados di Unix lo considerarono subito indegno di qualsiasi considerazione.

2.6 A new era

Ecco la situazione nel 1980: tre culture, simili ma organizzate intorno a diverse tecnologie. La cultura ARPAnet/PDP-10 sposata a LISP, MACRO, TOPS-10 e ITS. Il popolo di Unix e C, con i loro PDP-11, VAX e connessioni telefoniche di modesta entità. E infine un'orda anarchica di appassionati dei primi microcomputer, decisi a portare al popolo la potenza del computer.

Tra tutte queste, la cultura ITS poteva ancora rivendicare il posto d'onore. Ma sul Laboratorio si stavano addensando nubi minacciose. La tecnologia PDP-10 dipendente da ITS, cominciava a essere datata e il Laboratorio stesso era diviso in fazioni fin dai primi tentativi di commercializzazione della tecnologia AI. Alcune delle migliori menti del Laboratorio (e di SAIL e CMU), si erano lasciate attirare da lavori molto ben retribuiti presso società di nuova costituzione.

Il colpo di grazia fu inferto nel 1983, quando DEC cancellò la sua adesione al PDP-10 per concentrarsi sulle linee VAX e PDP-11. ITS non aveva più un futuro. In virtù della sua scarsa portabilità, infatti, l'idea di trasportarlo da un hardware all'altro era impensabile per chiunque. La variante funzionante su Unix di Berkeley VAX, divenne il sistema prediletto dagli hacker, e chiunque avesse rivolto lo sguardo al futuro, si sarebbe reso conto di quanto rapidamente crescesse la potenza dei microcomputer e con quale velocità avrebbero spazzato via tutto quello che li aveva preceduti.

Fu all’incirca in questo periodo che Levy scrisse Hackers. Una delle sue principali fonti di informazione fu Richard M. Stallman (inventore di Emacs), una figura chiave del Laboratorio e accanito oppositore della commercializzazione della tecnologia del Laboratorio.

Stallman (meglio conosciuto con le sue iniziali e login name, RMS), creò la Free Software Foundation, dedicandosi alla produzione di free software di alta qualità. Levy lo elogiò quale “ultimo vero hacker”, una descrizione che si rivelò fortunatamente errata.

Il grandioso progetto di Stallman riassunse chiaramente la transizione che subì la cultura degli hacker nei primi anni ’80: nel 1982 egli iniziò la costruzione di un intero clone di Unix, scritto in C e disponibile gratuitamente. Si può quindi dire che lo spirito e la tradizione di ITS furono preservati come parte importante della più nuova cultura hacker, incentrata su Unix e VAX.

Sempre in quello stesso periodo, la tecnologia dei microchip e della local area network iniziarono a fare presa sul mondo degli hacker. Ethernet e il microchip Motorola 68000 costituirono una combinazione teoricamente molto potente e solo dopo numerosi tentativi si arrivò alla prima generazione di ciò che oggi conosciamo come workstation.

Nel 1982, un gruppo di hacker Unix di Berkeley, fondò Sun Microsystems con la convinzione che Unix funzionante su un hardware con base 68000, relativamente economico, sarebbe stata la combinazione vincente per una grande varietà di applicazioni. La previsione si rivelò esatta e la loro intuizione definì il modello che l’intera industria avrebbe seguito. Sebbene i loro prezzi non erano ancora alla portata della maggior parte degli utenti, le workstation erano relativamente economiche per università e grandi aziende. Reti formate da questa nuova generazione di computer (uno per utente), sostituirono rapidamente gli ormai sorpassati VAX e altri sistemi time-sharing.

2.7 L’era del free Unix

Quando nel 1984 la AT&T iniziò ad essere svenduta e Unix divenne per la prima volta un prodotto commerciale, il mondo degli hacker si divideva in una “network nation”, relativamente coesiva e centrata su Internet e Usenet, in cui venivano per lo più usati minicomputer o workstation funzionanti con Unix, e una vasta ma disorganizzata “hinterland” di appassionati di microcomputer.

La classe di macchine workstation costruite da Sun e da altri, aprì nuovi

orizzonti agli hacker. Queste erano concepite per realizzare grafica di livello professionale e trasferire e gestire dati condivisi attraverso una rete. Nel corso degli anni ’80, il mondo degli hacker si mostrò attento alle sfide di software e strumenti per sfruttare al massimo queste caratteristiche. Il gruppo Unix di Berkeley sviluppò un supporto integrato per i protocolli ARPAnet che offriva una soluzione al problema delle reti favorendo un’ulteriore crescita di Internet.

Numerosi furono i tentativi di semplificare l’uso degli strumenti di grafica delle workstation. Il sistema che prevalse fu l’X Window System. Uno dei fattori che determinarono il suo successo fu dato dalla disponibilità dei suoi sviluppatori a fornire gratuitamente i sorgenti, secondo l’etica hacker, e a distribuirli tramite Internet. La vittoria di X sui sistemi di grafica proprietari (incluso quello offerto dalla stessa Sun), fu un’importante messaggio di cambiamento che, pochi anni dopo, avrebbe profondamente influenzato lo stesso Unix.

La rivalità tra ITS e Unix generava ancora qualche occasionale manifestazione di collera faziosa (per lo più proveniente dalla parte dei sostenitori dell’ex-ITS). L’ultima macchina ITS cessò comunque di funzionare per sempre nel 1990. I suoi partigiani si ritrovarono senza più un posto dove stare e furono in larga parte assimilati dalla cultura Unix non senza lamentele.

Nell’ambito degli hacker della rete, la grande rivalità negli anni ’80 era tra i sostenitori della versione Unix di Berkeley e quella di AT&T. Sono ancora oggi reperibili copie di un manifesto di quel periodo che riportava un combattente, in stile cartoon, con ali a forma di X, preso in prestito dal film Guerre Stellari, in fuga da una Death Star (stella morta) in esplosione contrassegnata dal logo AT&T. Gli hacker di Berkeley amavano vedersi come i ribelli contro i crudeli imperi aziendali. La versione Unix di AT&T non riuscì mai a competere sul mercato con il concorrente BDS/Sun, sebbene si aggiudicò la guerra degli standard. Nel 1990, le versioni AT&T e BSD divennero difficili da distinguere avendo l’una adottato molte innovazioni dell’altra e viceversa.

Agli inizi degli anni ’90, la tecnologia delle workstation del decennio precedente cominciava a vedersi seriamente minacciata da nuovi personal computer, a basso costo e dalle alte prestazioni, basati sul chip Intel 386 e i suoi discendenti.

Per la prima volta, ogni singolo hacker poteva finalmente permettersi di disporre anche a casa di macchine paragonabili, per potenza e capacità di

memoria, ai minicomputer di un decennio prima, macchine Unix in grado di supportare un ambiente di sviluppo completo e di comunicare con Internet.

In questo nuovo scenario, il mondo MS-DOS rimase beatamente allo scuro degli sviluppi in corso. Nonostante le fila degli appassionati di microcomputer della prima ora si ingrandirono rapidamente fino a diventare una popolazione di hacker DOS e Mac di dimensioni ancora maggiori rispetto alla cultura “network nation”, essi non riuscirono mai a sviluppare una cultura consapevole. Il ritmo dei cambiamenti era talmente veloce che numerose diverse culture tecniche nacquero e cessarono di esistere con la rapidità di una farfalla, senza mai raggiungere la stabilità necessaria allo sviluppo di un gergo, di un folklore e di una storia propri. L’assenza di una rete realmente pervasiva, paragonabile a UUCP o a Internet, non permise loro di diventare una network nation. Il crescere degli accessi a servizi commerciali online, come CompuServe e Genie, ma parallelamente la non diffusione in bundle di strumenti di sviluppo per sistemi operativi non-Unix, significava poco materiale su cui lavorare. Questa situazione impedì lo svilupparsi di una tradizione di collaborazione tra gli hacker.

La corrente hacker più importante, (dis)organizzata intorno a Internet, e finora largamente identificata con la cultura tecnica di Unix, non era interessata ai servizi commerciali. I suoi adepti volevano solo strumenti migliori e più Internet, cose che l’economico PC a 32-bit promise di mettere alla portata di tutti.

Ma dov’era il software? Le macchine Unix commerciali restavano comunque costose. Nei primi anni ’90, numerose società fecero una prova vendendo porting di Unix BDS o AT&T per macchine PC. Il successo si rivelò elusivo, i prezzi non erano scesi di molto e (ancora peggio) non si ottenevano sorgenti modificabili e ridistribuibili per il proprio sistema operativo. Il tradizionale modello di software-business non stava affatto fornendo agli hacker ciò che volevano.

Neanche con la Free Software Foundation la situazione migliorò. Lo sviluppo di HURD, il tanto sospirato kernel Unix gratuito per hacker promesso da RMS, rimase fermo per anni e non riuscì a produrre alcunché di utilizzabile fino al 1996 (sebbene dal 1990 la FSF avesse fornito quasi tutti gli altri complicati componenti di un sistema operativo simile a Unix).

Ciò che dava davvero motivo di preoccupazione era che, con l’inizio degli anni ’90, si cominciava a vedere con chiarezza come dieci anni di tentativi

di commercializzare Unix stessero dopotutto fallendo. La promessa di Unix, di rendere portabili le cross-platform si perse tra mezza dozzina di versioni proprietarie di Unix. I detentori di Unix proprietario diedero prova di tanta lentezza e inettitudine nel campo del marketing, che Microsoft fu in grado di inglobare la maggior parte della loro fetta di mercato con la tecnologia del sistema operativo Windows, incredibilmente inferiore a quella Unix.

Nei primi mesi del 1993, qualsiasi osservatore pessimista avrebbe avuto tutti i motivi per decretare l'imminente fine della storia di Unix e della fortuna della sua tribù di hacker, cosa tra l'altro predetta sin dai tardi anni '70 a intervalli regolari di 6 mesi.

In quei giorni, era pensiero comune la fine dell'era del tecno-eroismo individuale e che l'industria del software e la nascente Internet sarebbero state dominate da colossi come Microsoft. La prima generazione di hacker Unix sembrava invecchiata e stanca (il gruppo di Ricerca della Scienza Informatica di Berkeley chiuse i battenti nel 1994). Il periodo non era tra i più felici.

Fortunatamente, ci furono cose che sfuggirono all'attenzione della stampa specializzata e perfino alla maggior parte degli hacker, cose che avrebbero prodotto sviluppi positivi verso la fine del 1993 e l'inizio del 1994.

In futuro, questa situazione avrebbe portato la cultura a imboccare una strada completamente nuova, disseminata di insperati successi.

2.8 I primi free Unix

Dal gap provocato dal fallimento dell'HURD, era emerso uno studente dell'Università di Helsinki di nome Linus Torvalds. Nel 1991, cominciò a sviluppare un kernel free Unix per macchine 386 usando un kit di strumenti della Free Software Foundation, in particolare partì estendendo Minix con un modulo per la comunicazione dei processi. Il suo rapido successo nella fase iniziale, attrasse molti hacker di Internet, volonterosi di aiutarlo nello sviluppo del suo Linux, una versione Unix con sorgenti interamente free e redistribuibili.

Anche Linux aveva i suoi concorrenti. Nel 1991, contemporaneamente ai primi esperimenti di Linus Torvald, William e Lynne Jolitz stavano sperimentando il porting di Unix BSD sul 386. La maggior parte di coloro che paragonavano la tecnologia BSD agli sforzi iniziali di Linus, si aspettavano che i porting di BSD diventassero i più importanti free Unix su PC.

La caratteristica fondamentale di Linux, tuttavia, non era tanto tecnica

quanto sociologica. Fino allo sviluppo di Linux, era pensiero comune che qualsiasi software complicato come un sistema operativo, dovesse essere sviluppato in modo attentamente coordinato da un ristretto gruppo di persone ben collegate tra di loro. Questo modo di operare era, ed è tuttora, tipico sia del software commerciale che delle grosse cattedrali di freeware costruiti dalla Free Software Foundation negli anni ’80; così come dei progetti freeBSD/netBSD/OpenBSD, che allargarono il campo di applicazione del porting originale 386BSD dei Jolitz.

Linux si evolse in modo completamente differente. Fin quasi dalla sua nascita, fu casualmente “preda di hacking” da parte di un vasto numero di volontari collegati solo tramite Internet. La qualità fu mantenuta non da rigidi standard o autocrazia, ma dalla strategia semplice e naive di proporre settimanalmente delle idee e di ricevere opinioni in merito da centinaia di utenti ogni giorno, creando una sorta di rapida selezione darwiniana sulle modifiche introdotte dagli sviluppatori. Con stupore da parte di quasi tutti, il progetto funzionava piuttosto bene.

Verso la fine del 1993, Linux fu in grado di competere per stabilità e affidabilità, con molti Unix commerciali, ospitando una grande quantità di software. Esso stava perfino cominciando ad attirare il porting di applicazioni software commerciali. Un effetto indiretto di questo sviluppo, fu lo spazzare via la maggior parte dei piccoli fornitori di Unix commerciali - la loro caduta fu anche determinata dalla mancanza di hacker e potenziali utenti ai quali vendere. Uno dei pochi sopravvissuti, BSDI (Berkeley System Design, Incorporated), fiorì offrendo sorgenti completi, con il suo Unix base BSD, e coltivando stretti legami con la comunità hacker.

All’epoca tali sviluppi non furono pienamente rilevati dalle comunità hacker e non lo furono affatto al di fuori di essa. La tradizione hacker, a dispetto delle ripetute predizioni su una sua imminente fine, stava proprio iniziando a riorganizzare il mondo del software commerciale a propria immagine. Trascorsero ancora cinque anni prima che questa tendenza iniziasse a palesarsi.

2.9 La grande esplosione del Web

L’iniziale crescita di Linux coincise con un altro fenomeno: la scoperta di Internet da parte del grande pubblico. I primi anni ’90 videro l’inizio di una

fiorente industria dell'Internet provider, che forniva connessioni al pubblico per pochi dollari al mese. Dopo l'invenzione del World Wide Web, la già rapida crescita di Internet accelerò a rotta di collo.

Nel 1994, anno in cui il gruppo di sviluppo Unix di Berkeley chiuse ufficialmente i battenti, molte diverse versioni di free Unix (Linux e i discendenti del 386BSD) catalizzarono l'interesse degli hacker. Linux era distribuito su CD-ROM, e andava via come il pane. Alla fine del 1995, le maggiori aziende informatiche cominciarono a promuovere i propri hardware e software giocando la carta della loro grande compatibilità con Internet!

Nella seconda metà degli anni '90, l'attività degli hacker si incentrò sullo sviluppo di Linux e sulla diffusione di massa di Internet. Il World Wide Web era riuscito a trasformare Internet in un mezzo di comunicazione di massa, e molti hacker degli anni '80 e '90, intrapresero l'attività di Internet Service Provider fornendo accesso a questo nuovo mondo.

La diffusione di massa di Internet, aveva perfino portato la cultura hacker ad essere rispettata in quanto tale. Nel 1994 e 1995, l'attivismo hacker fece naufragare la proposta Clipper che avrebbe posto sotto il controllo del governo un metodo di codifica. Nel 1996, gli hacker si mobilitarono per sconfiggere il "Communications Decency Act" (CDA), e scongiurare il pericolo di censura su Internet.

"Breve storia sugli hacker", Eric Raymond (1996).

2.10 Il nocciolo

La cultura del Free Software sostenuta dagli hacker si pone all'attenzione del grande pubblico, e quindi del mercato, grazie al sistema operativo Linux. Linux verrà descritto nel capitolo 4, qui si vuole delineare la storia della sua creazione.

Linus Torvalds, finlandese, ventenne, studente alla facoltà di informatica di Helsinky. Nel 1990, dovendo studiare sul testo di Andrew Tannenbaun "Operating Systems: Design and Implementation", si procura un PC e installa Minix, un piccolo sistema operativo simile a Unix creato a fini didattici. Incomincia ad esercitarsi programmando un rudimentale task switching, lo estende per riuscire a leggere i newsgroup da casa. Ha bisogno di scaricare dei file dalla rete, per salvarli sul suo Pc ha bisogno di gestire il disco, pen-

sa [9]: “Se incorporo la gestione del disco nel kernel dovrò riscrivere tutto quando cambio PC, se scrivo un device driver lo potrò sostituire”.

Scrisse i device driver fondamentali: per la tastiera, per il monitor, per il modem, per il disco rigido.

Lo stesso Torvalds: “Quando hai un task-switching, un file system, i device driver allora hai Unix”.

Mise a disposizione il suo lavoro, che venne chiamato “Linux” come il suo nome di lavoro, su un server FTP all’Università di Helsinky affinché chi studiava Minix ci potesse dare un’occhiata.

Nel Gennaio del 1992 circa un centinaio di programmatori stavano usando Linux, ma fu un momento cruciale: chi trovava un errore inviava la patch a Torvalds che si occupava di integrarla nel ramo principale del codice. Linux incominciò a crescere in robustezza e funzionalità. Essendo i codici sorgenti liberamente disponibili il gruppo di programmatori cresceva espandendosi per tutta la Terra. I più anziani, come collaborazione, divennero i coordinatori delle aree di sviluppo.

Linux era ormai un sistema operativo stabile ma fondamentalmente fine a sé stesso: essendo nato come studio non aveva applicativi.

2.11 La polpa

Dal 1984, a Boston, Richard Stallman lavorava sul progetto GNU (acronimo recursivo per “GNU’s not Unix”), il tentativo di creare un intero ambiente (dal sistema operativo agli applicativi) basato su software libero.

Stallman era intenzionato a incominciare dal sistema operativo, non trovando qualcosa di utilizzabile (cioè libero) in giro, decise di scriverlo per conto suo. Quindi serviva un compilatore. Quindi serviva un editor di testi. Scrisse Emacs, l’editor tutto fare dei sistemi Unix. Scrisse GCC, il compilatore più usato nel mondo. Questi progetti e i relativi sviluppi, dovuti al successo che incontrarono e al fatto di essere completamente compatibili con i sistemi Unix esistenti, ritardarono il progetto GNU di qualche anno. Quando Stallman si accinse a gestire lo sviluppo del sistema operativo (ormai poteva contare su un buon numero di collaboratori) varò il progetto HURD: la creazione del nuovo sistema operativo. Fu deciso di utilizzare il microkernel Mach sviluppato prima dalla Carnegie Mellon University e poi dall’Università dello Utah. HURD era un gruppo di server che girava su Mach svolgendo le funzio-

ni del kernel Unix. Le difficoltà furono subito grandi: il debugging dei server multi-thread che si scambiano messaggi si rivelò estremamente complesso.

Ancora oggi HURD non è pronto al rilascio.

2.12 Il frutto

Stallman aveva a disposizione però, dal 1992, un altro sistema operativo rispondente ai canoni del Free Software: Linux. D'accordo con Torvalds lavorò alla non facile integrazione degli applicativi GNU con Linux. Nacque un ambiente stabile e ricco di funzionalità: GNU/Linux.

Quando ci si riferisce oggi a Linux in realtà si intende la distribuzione di software GNU/Linux, dove Linux è il sistema operativo e GNU la risma di programmi che permettono l'esecuzione di un gran numero di task. Sono presenti editor, compilatori, interfacce utente, device drivers, programmi di grafica e musicali, giochi, suite di office automation, ecc. ecc.

L'esistenza di GNU/Linux dimostrò la reale possibilità di produrre codice stabile, performante, a ogni livello di complessità con il libero scambio del codice sorgente.

Quando Netscape dovette elaborare una strategia per non uscire dal mercato decise di seguire la via del Free Software.

2.13 Nasce la "Open Source Iniative"

Raymond ricorda che l'etichetta "Open Source" venne fuori la prima volta nel 1998 a Palo Alto durante una riunione di alcuni promotori del Free Software [10]. Si discuteva come reagire all'annuncio di Netscape riguardante la distribuzione dei sorgenti del Navigator. Raymond stesso era stato invitato dalla Netscape a collaborare come consulente strategico dell'iniziativa. Raymond aveva conquistato l'attenzione della Netscape come conoscitore del movimento hacker e per aver redatto uno studio, "The Cathedral and the Bazaar", in cui esaminava il metodo di produzione a sorgenti liberi.

Tutti i partecipanti capirono che si era presentata l'occasione per spiegare al mondo commerciale l'intrinseca superiorità dei processi a sviluppo aperto e che era finito il tempo del muro contro muro tra software libero e software

commerciale. Le distinzioni ideologiche non erano necessarie alla causa, serviva affrontare la questione da un punto di vista pragmatico poiché persone pragmatiche erano quelle da convincere.

Il termine “Open Source” fu propagandato con tutti i mezzi dalla stampa al web. Chi si assunse la responsabilità di diventare il porta bandiera dell’iniziativa fu lo stesso Raymond. Venne elaborata una campagna di informazione rivolta al pubblico e a interessare le grandi società di finanziamento. L’obbiettivo fu centrato in entrambi i contesti: oggi Open Source è un termine in voga presso chi si occupa di informatica e i “venture capital”, cioè l’iniziative di finanziamento atte a procacciare alti guadagni, stanno entrando massicciamente nella produzione di software “Open Source”.

Il 31 Marzo 1998 il codice sorgente del Netscape Navigator venne rilasciato. Raccolse un immediato consenso da parte della comunità hacker sotto forma di correzioni, migliorie e nuove funzionalità.

Da allora la lista delle società aderenti al nuovo modello di sviluppo è diventata lunghissima. Possiamo citare i casi più significativi:

- La Corel Computer Corporation (Maggio 1998) annuncia l’uscita di Netwinder, un network computer economico con Linux come sistema operativo nativo.

- La Corel (Maggio 1998) annuncia di voler portare Word Perfect e l’intera sua suite da ufficio su Linux.

- Sun Microsystems e Adaptec (Maggio 1998) aderiscono alla Linux International garantendo il supporto a Linux nei loro prodotti software e hardware.

- Oracle e Informix (Luglio 1998) annunciano l’intenzione di portare i propri database su Linux

- Sun Microsystems (Agosto 1998) rilascia il sorgente di Solaris.

- SCO (Agosto 1998) annuncia che sta già lavorando alla piena compatibilità tra la propria versione di Unix e Linux.

- Intel e Netscape (Settembre 1998) acquistano quote di minoranza di Red Hat, società distributrice di Linux leader di mercato in America.

- La catena di negozi Jay Jacobs (Novembre 1998) decide di avvalersi, per i punti vendita, di una rete basata su Linux.
- HP e SGI (Gennaio 1999) annunciano lo stesso di giorno di voler supportare Linux sulle proprie macchine.
- IBM (Febbraio 1999) annuncia di voler supportare Linux sulle proprie macchine, di rilasciare una versione di Lotus per Linux e la creazione di una partnership con Red Hat.
- Apple (Marzo 1999) rilascia il codice sorgente di Darwin (il nucleo del sistema operativo di MacOSX).
- Microsoft (Giugno 1999) dichiara che le distribuzioni Linux vendono più di Windows 98.
- Amiga (Luglio 1999) annuncia che Linux sarà il sistema operativo della prossima generazione Amiga Operating Environment.

L'elenco da qui in poi si infittisce ma spiccano due casi eclatanti: la NASA (l'Agenzia Aerospaziale Americana) per garantirsi la massima sicurezza e capacità d'intervento decide di comprare solo software corredato dei sorgenti, i Fermi Labs di Chicago decidono di costruire la nuova rete dedicata al super calcolo usando il kernel Linux.

2.14 Febbraio 2000

Al momento in cui viene redatto questo documento rimane sostanzialmente invariata la fase di espansione e di conferma per l'Open Source (OS).

Catene commerciali come Ikea o industrie come Boeing si sono interamente affidate a Linux per la gestione interna. Crescono le società di consulenza specializzate. OS è in costante crescita sul mercato dei server di rete sia in Internet, dove occupa mediante il web server Apache il 53

Si registrano tassi di crescita vertiginosi per le società, tra cui Red Hat, SuSe, Caldera in primis, che vendono i supporti contenenti interi ambienti basati su Linux. La distribuzione SuSe è costituita da sei cd-rom per un totale di più di 1500 programmi e costa meno di 50 dollari. Con un solo prodotto si viene in possesso di tutto l'esistente (o quasi) relativo a Linux: decine

di compilatori, la suite per ufficio della Corel, le interfacce grafiche KDE e GNOME, programmi di grafica, di editing, database, linguaggi di scripting, giochi, moduli dedicati dal palmare al server di rete. Gli acquirenti sono milioni, tale cifra dimostra che Linux è ormai un fenomeno di mercato, se poi si considera che si tratta di distribuzioni libere la quota di utenti raggiunta è ancora maggiore.

I maggiori limiti all'espansione di Linux erano costituiti dal fatto che poteva risultare ostico da installare e mancava di un'interfaccia user-friendly. La barriera delle difficoltà di installazione è stata abbattuta dai distributori sopra menzionati: per raggiungere quote di mercato più ampie hanno sovvenzionato lo sviluppo di tool di installazione user-friendly. Oggi installare Linux è operazione di mezz'ora, se si incontrano problemi sono disponibili linee telefoniche di assistenza.

Le interfacce grafiche, GNOME e KDE, non hanno ancora raggiunto la piena completezza ma esistono versioni stabili altamente versatili. Ho avuto modo di constatare un tasso di errore sul livello dei prodotti Microsoft, con la differenza che le interfacce OS sono ancora in fase di sviluppo!

Le compagnie distributrici più altre come la VA Research sovvenzionano lo sviluppo OS stipendiando programmatori. Quale sia il ritorno in termini economici nella creazione di software che è gratuitamente a disposizione di tutti sarà spiegato nel prossimo capitolo.

Facilità di installazione insieme a interfacce grafiche vuol dire accedere al mercato dei desktop, ed è questa l'ultima sfida di Linux: invadere il mercato di proprietà Microsoft.

2.15 Prospettive

È opinione comune degli analisti di mercato che Linux si imporrà definitivamente nel mercato dei server.

Per quanto riguarda la fascia desktop molto dipende dalla diretta rivale Microsoft.

Microsoft 2000 è appena stato rilasciato ed è troppo presto per darne una valutazione. Considerando che un anno e mezzo fa era un "mostro" da sei milioni di righe di codice in costante aumento è difficile credere che sia totalmente sicuro e stabile; inoltre le prime beta evidenziarono pesanti lacune nella sicurezza. È possibile che la Microsoft abbia rimediato agli errori, ma

è anche possibile che alla fine Windows 2000 non riesca a imporsi definitivamente su tutti i mercati in cui entrerà. Comunque sia il potenziale di risorse della Microsoft è tale che se anche Microsoft 2000 si rivelasse il più grosso fallimento commerciale del secolo ciò non intaccherebbe, a tempi brevi, il patrimonio della casa di Redmond.

Un'altra incognita, sempre riguardante Microsoft, è rappresentata dalla causa antitrust in corso. Potrebbe risolversi con lo smembramento in società più piccole (come già avvenne per AT&T) oppure potrebbe risolversi, come si vocifera nei newsgroup dei bene informati, nel rilascio dei codici sorgenti. Alcune indiscrezioni indicano la seconda via come la più possibile, ma ancora nulla di certo è trapelato. Se veramente la Microsoft rilasciasse i sorgenti avremmo a disposizione uno dei più grandi patrimoni informatici del mondo. A quel punto perderebbe senso la faziosità Microsoft vs. Linux, tanto cara agli Hacker di vecchia memoria, per un più esteso welfare del software.

In attesa dell'epilogo delle vicende giudiziarie di Microsoft possiamo valutare che non ci sarà, in tempo breve (diciamo per tutto il 2000), un reale scontro sul mercato desktop. Per Linux ci sarà spazio per la competizione solo in caso di espansione del mercato. Imprenditori lungimiranti e preparati potrebbero scegliere di puntare su OS, ma dove Microsoft è già presente sarà impossibile scalzarla se non per quote risibili. L'imponente apparato commerciale di Microsoft ha facilmente ragione delle ancora piccole strutture di supporto per OS, inoltre ormai gli utenti sono abituati all'ambiente Windows e, per certi settori, passare anche solo a un'interfaccia diversa costituerebbe un problema.

In tempi più lunghi Linux guadagnerà progressivamente quote, questo sarà dovuto al fatto che l'utenza media accetterà con meno sospetto la filosofia OS potendone constatare i risultati a costi nulli. Linux è disponibile in tutte le edicole, di conseguenza nuove utenze si abitueranno a un'interfaccia diversa dalla solita e incominceranno a richiederla. Aumentando la domanda di servizi aumenterà l'offerta dei prodotti e qui si innescherà il circolo virtuoso tipico dell'Open Source: sviluppo sicuro in tempi rapidi.

A quel punto si avrà una netta contrazione del mercato dei sistemi proprietari, avendo dalla loro solo "l'abitudine del cliente".

Un problema ben più grave che potrebbe diventare il vero ostacolo alla produzione e, soprattutto, al miglioramento delle risorse informatiche è rappresentato dai tentativi a livello giuridico di permettere il brevetto sul soft-

ware. Una trattazione esaustiva sull'argomento esula dallo scopo di questa tesi e meriterebbe un lavoro interamente dedicato. Qui si vuole appuntare l'impatto che questa infausta decisione potrebbe avere sullo sviluppo di risorse tecnologiche a libero accesso.

In primo luogo esiste un'intrinseca difficoltà nel risalire all'autore di un algoritmo, o di discernere l'entità effettiva dei diversi contributi. Quindi l'attribuzione di un brevetto sul software comporta notevoli difficoltà sul piano pratico, queste difficoltà rallenterebbero lo sviluppo, con spreco di tempo e risorse, e di conseguenza aumenterebbero i costi.

In secondo luogo i brevetti potrebbero facilmente diventare strumento di ritorsione in mano alle compagnie con più risorse contro compagnie più piccole, con meno fondi ma maggiori innovazioni (caso comune nell'informatica). Brevettare una propria invenzione ha costi accessori non indifferenti. Una volta ottenuto il brevetto è poi necessario difenderlo da sfruttamenti non consentiti. Difendere un brevetto comporta aprire una causa legale, assumere il patrocinio da parte di avvocati specializzati, partecipare a un processo che, per la "globalizzazione del mercato", è molto probabile venga tenuto in un paese lontano dal proprio.

Sviluppare nuova tecnologia diventerà molto rischioso: rilasciare un prodotto che incappa poi nell'effrazione di un qualche oscuro brevetto software già registrato potrebbe frustrare sforzi e ricerche di anni, oltre che essere economicamente rovinoso.

Il rilascio di brevetti sul software gioverà unicamente a chi può contare su risorse tali da potersi permettere personale dedicato alla gestione dei brevetti propri e altrui. Cioè si avrebbe un ulteriore strumento in mano di chi è già massicciamente presente sul mercato, non di chi sta emergendo.

I brevetti nascono con la nobile intenzione di proteggere l'inventore, in realtà poi la pesante burocrazia inerente la gestione dei brevetti danneggia l'inventore stesso.

Il Free Software verrebbe pesantemente penalizzato dalle legislazioni sui brevetti software perché i brevetti colpiscono proprio la principale risorsa dello sviluppo a sorgenti aperti: la condivisione dell'informazione. I brevetti vincolerebbero programmatori e programmi a pesanti restrizioni. Il brevetto, e quindi il segreto industriale, impedirebbero discussioni e scambi di conoscenze. Inoltre sarebbe altamente rischioso costruire un programma OS che copre le funzionalità di un programma proprietario: il rischio di utilizzare algoritmi

brevettati sarebbe molto grande, troppo grande per chi non è supportato da adeguate strutture.

OS ha dimostrato di potersi imporre senza l'appoggio di grandi strutture commerciali (a beneficio dei costi). Proprio l'assenza di queste strutture permetterebbe a compagnie più "organizzate" di estendere brevetti su prodotti OS, previo leggere modifiche.

Negli Stati Uniti è da poco entrata in vigore una normativa analoga nonostante la forte opposizione delle piccole società e dei sostenitori del Free Software. Il primo risultato è stata la nascita di società di informatica senza neanche un tecnico tra i dipendenti, solo uffici legali. Queste società coprono con brevetto tutto il software che riescono a trovare senza un possessore che ne rivendichi la paternità. Lo scopo di queste azioni è possedere il brevetto nel momento in cui qualcuno decida di usare, o riutilizzare, quel software per poter vendere l'usufrutto del brevetto.

2.16 Hackers vs Crackers

È importante distinguere gli hacker dai cracker. Sono innanzitutto due comunità diverse. Anche se i giornali riportano il termine hacker in qualsiasi contesto. La differenza tra i due gruppi è palese: gli hacker creano, i cracker distruggono. Questo indipendentemente dal fatto che entrambi siano in grado di individuare le falle nei sistemi di sicurezza e di sfruttarle. Mentre per gli hacker è fondamentale non danneggiare il sistema "visitato", i cracker fanno del danno la loro principale attività.

Le due comunità si gestiscono in modo differente. Per gli hacker, come già scritto, la conoscenza è un patrimonio da condividere; i cracker si comportano diversamente: si scambiano i binari dei programmi "killer", come prova d'abilità, ma sono "gelosi" riguardo le conoscenze alla base del programma. Questo spiega anche perché la comunità cracker segna il passo rispetto alla comunità hacker: non essendoci un reale scambio di informazioni continuano a inventare la ruota e il fuoco.

Obiettivamente si spera che i cracker non adottino una mentalità Open Source.

Un grosso regalo che viene dall'America

I legislatori americani hanno fatto un bel regalo alla Comunità Europea, notoriamente in svantaggio nel campo dello sviluppo di nuove tecnologie. Con la legislazione sul brevetto del software gli sviluppatori americani sono imbrigliati in una ragnatela di norme e di processi giudiziari tale da rallentare notevolmente le loro intrinseche capacità di innovazione. È come se avessero deciso di correre la corsa dell'innovazione tecnologica indossando scarpe di piombo.

La Comunità Europea potrebbe approfittare notevolmente della situazione. In primo luogo evitando di commettere lo stesso errore, cioè non permettendo di coprire con brevetto il software. In secondo luogo applicando una legislazione che protegga la metodologia di sviluppo OS.

Il software non è una scienza pura, ma trattato come tale, con la piena disponibilità del sorgente, assurge livelli di sviluppo rapidissimi. Livelli di sviluppo necessari a colmare il gap tecnologico con America e Giappone. Se poi decidessimo addirittura di scavalcare i nostri maestri di sempre potremmo tentare la strada dell'Open Hardware, cioè applicare i principi Open Source alla produzione dei circuiti integrati: il sorgente dell'hardware sarebbe il progetto del chip! E correre così con due cavalli: l'hardware e il software.

3 Sviluppo Open Source e Applicazione Industriale

3.1 Introduzione

Nel capitolo precedente si sono descritte le motivazioni sociali del lavoro volontario su cui si basa lo sviluppo del Free Software. In questo capitolo si vuole descrivere la metodologia su cui si base tale sviluppo per poi rispondere alla domanda: è possibile organizzare industrialmente lo sviluppo di software secondo il modello Free Software? Si vuole chiarire se i risultati finora ottenuti dal movimento Open Source e Free Software sono frutto di particolari condizioni fortunate o possono essere riprodotti in un'economia di scala.

Verrà esaminato il caso Linux, esempio massimo di successo nel mondo Open Source, e sarà comparato ad altri progetti analoghi per valutare le cause di buona riuscita o di fallimento.

Poiché il metodo seguito da Torvalds in Linux non è l'unico possibile seguirà la descrizione di procedimenti alternativi (legati ai progetti Apache e Perl) che hanno avuto successo.

Esposti i metodi di produzione Open Source si cercherà di valutarne la validità commerciale, ossia si risponderà alla domanda: "Quando, e come, un progetto Open Source può essere economicamente profittevole?"

Saranno di grande aiuto gli scritti di Eric Raymond "The Cathedral and the Bazaar" [11], per analizzare l'impostazione di un progetto Open Source di successo, e "The Magic Cauldron", per valutare la validità commerciale dei progetti Open Source [12].

3.2 Il Gigante

Nel capitolo riguardante la storia del Free Software si è narrato di uno studente finlandese, Linus Torvalds, che avvia e gestisce il progetto di un sistema

operativo, Linux, in grado di soppiantare i concorrenti Unix e di porsi come il maggiore contendente di mercato dei prodotti Microsoft.

La creazione di un sistema operativo è un'operazione molto complessa, ritenuta da sempre competenza di team di sviluppo ristretti, affiatati e sapientemente gestiti. Inoltre essendo un processo lungo e altamente specializzato richiede l'uso di risorse a disposizione solo di grandi investitori.

Torvalds sembra avere smentito i canoni della produzione industriale: non aveva un progetto in senso compiuto, non aveva fondi ed è partito solo.

Il risultato, Linux, è esperibile da tutti. Come è potuto capitare? O meglio: come ha fatto Torvalds a raccogliere centinaia di super tecnici volontari? Come è riuscito a organizzarli? Come ha gestito un progetto che nell'arco di dieci anni, a tappe incalzanti ma senza una programmazione delle release, è evoluto da sistema operativo per studenti a piattaforma completa multi uso?

Procedendo con metodo sarà possibile fornire un'interpretazione sufficientemente chiara del fenomeno Linux.

Innanzitutto chiariamo che Torvalds non partì , nel 1990, con l'intenzione di costruire un sistema operativo che soppiantasse i concorrenti in ambito commerciale. Fu l'opera di una serie di felici intuizioni sapientemente gestite a portare al compimento un'opera appena abbozzata. Il fatto notevole è che tali intuizioni possono essere considerate un metodo nello sviluppo di software.

È necessario considerare che Torvalds non concepì un progetto originale ma utilizzò il sistema operativo Minix, una versione molto semplificata di Unix ad uso didattico. Quando mise a disposizione il suo lavoro ai colleghi universitari poté così contare su una generale buona conoscenza del codice base. Oggi Linux non ha più niente del codice originario di Minix, ma aver potuto contare su una knowledge base comune ha ampliato il bacino d'utenza.

Ricordando il modello sociale basato sul dono, spiegato nel capitolo precedente, Torvalds, offrendo un qualcosa di funzionante e comprensibile ai più si è messo nella condizione del gestore della fonte. Un sistema operativo è un'opera altamente complessa, fornendo un qualcosa di funzionante, comprensibile e estensibile, per nulla completo, Torvalds ha offerto un ampio spazio nella noosfera legata al progetto.

Alla creazione di Linux hanno partecipato hobbisti, studenti e tecnici specializzati che hanno trovato l'occasione per valorizzare e dimostrare le proprie capacità e nel progetto hanno trovato lo spazio necessario alla propria

creatività.

È stato importante fornire un nucleo di partenza per almeno due motivi: in primo luogo dimostrare che era possibile e quindi aumentare la fiducia nella buona riuscita; in secondo luogo bisogna considerare che il codice era l'unica base di comunicazione, di conseguenza era necessario come veicolo per trainare lo sviluppo.

Questa è stata la prima fortunata intuizione: mostrare una via praticabile e accessibile.

Creato il gruppo occorreva tenerlo unito. Se si considera che dopo un anno si potevano contare un centinaio di attivi contribuenti, che dopo due anni erano alcune centinaia, che negli anni successivi si sono contati qualche migliaio di programmatori il rischio che la comunità si spaccasse per percorrere direzioni diverse nello sviluppo o che si perdesse il riferimento del codice comune su cui lavorare era più che tangibile. Ancora oggi un grande problema dei team distribuiti è il riferimento al codice di sviluppo: il rischio di modificare il lavoro altrui o di preparare versioni non compatibili è altissimo e causa di fallimento.

Come gestire allora un team sparso per il pianeta senza vincoli di dipendenza al suo interno? Torvalds seguì la strada della delega e del giusto riconoscimento. Man mano che il gruppo si estendeva le patch da esaminare sarebbero state troppe per un uomo solo, ma tuttavia c'era la tendenza, da parte degli sviluppatori più giovani, a riferirsi per consigli e revisioni agli sviluppatori che da più tempo erano impegnati con Linux. Gli sviluppatori più anziani diventarono i collettori del codice sviluppato e, di conseguenza, organizzavano il lavoro e raccoglievano le idee all'interno del loro ambito. L'architettura modulare di Linux, presente fin dalla nascita, facilitò questo processo.

Torvalds si dimostrò sempre attento nella gestione dei riconoscimenti per i contributi e nel valutare proposte e idee nuove. E si può dire furono queste le due sue più grandi qualità: l'incentivazione degli sviluppatori mediante il riconoscimento dei meriti e il saper distinguere le idee vincenti e applicarle.

Inoltre la complessità dell'intero lavoro e il carisma di Torvalds, sempre meno attivo come programmatore ma sempre più attento nell'organizzazione, rendevano la possibilità di biforcazione del codice molto remota. Prendere copia dei sorgenti e lanciare un proprio progetto sarebbe stato considerato pazzia dalla comunità hacker. Chi avrebbe seguito i secessionisti? Come avrebbero potuto garantire lo sviluppo di un progetto che già stava andando

molto bene?

In realtà oggi esistono una dozzina di diversi kernel Linux, ma queste "biforcazioni" del codice si sono presentate per rispondere a necessità di specializzazione. Abbiamo versioni di Linux real-time, per computer palmari, per reti di super calcolo, per workstation grafiche, più ovviamente ai kernel compatibili con le diverse famiglie di processori.

Un effetto della distribuzione remota dei programmatori fu la necessità di scambiare idee, progetti, codice unicamente per via e-mail e via ftp. Non potendo comunicare in altro modo se non scrivendo si rese necessaria una certa omogeneità stilistica per ovvie ragioni di praticità. Questi fatti spinsero i programmatori a scrivere documentazione di alta qualità. Oggi Linux è il sistema operativo meglio documentato. È sorto addirittura un gruppo, il "Linux Documentation Group", con il preciso scopo di scrivere e raccogliere la documentazione riguardante Linux. Sono disponibili le guide pratiche (i famosi "HOWTO-"), i manuali di specifiche tecniche, le guide utente. La necessità di comunicare ha vinto l'abituale reticenza dei programmatori a documentare il proprio lavoro. Risultato non indifferente raggiunto da Torvalds.

Dato un gruppo di sviluppatori motivati e organizzati occorre renderli produttivi.

Un impegno costante di Torvalds fu il continuo rilascio di versioni, in alcuni periodi anche più al giorno. Le idee che funzionavano venivano immesse nella distribuzione principale di Linux e rilasciate al pubblico il prima possibile. Metodo inusitato a tutto il mondo software. In genere i rilasci delle nuove versioni avvengono a distanza di almeno sei mesi e solo dopo un accurato lavoro di testing da parte del team di sviluppo. La nuova versione dovrebbe essere il più vicino possibile alla condizione "Error Free" e resistere all'impatto dell'utenza.

Torvalds ha ribaltato la tradizionale gestione dei rilasci. Per ogni significativa modifica metteva a disposizione una nuova versione da provare e testare. Gli utenti, molti dei quali programmatori, si accanivano nella caccia degli errori e dei possibili miglioramenti. Ben presto divenne palese che chi trovava l'errore e chi lo correggeva solo saltuariamente erano la stessa persona, nella maggior parte dei casi chi notava l'anomalia non era in grado di correggerla e chi era in grado di correggerla non incappava nell'anomalia. La fitta rete di comunicazione rendeva noti gli errori con esaurienti descrizioni e nel giro di ore giungevano le prime patch di correzione. La fortuna di Torvalds

non fu tanto l'avere utenti che fossero programmatori ma quanto l'intuizione di trattare tutti gli utenti al pari dei programmatori. Il lavoro di debugging non necessita di particolare organizzazione: può essere eseguito in parallelo, quindi perché non farlo fare a tutti?

Sottoponendo il codice a pubblica revisione spinse i programmatori a produrre codice leggibile e solido. Il fatto di essere continuamente al banco di prova sotto centinaia di occhi contribuì a prevenire la distribuzione di codice poco consistente. In questo modo Torvalds rilasciava frequentemente codice seriamente testabile.

Questo sistema si rivelò di straordinaria efficacia in termini di rapidità di sviluppo e di stabilità del codice. Se si considera che Linux è partito a metà 1990 e che nel 1992 costituiva già una piattaforma sufficientemente stabile per i prodotti del progetto GNU, nel 1993 veniva distribuito pronto da installare per un'utenza mediamente informatizzata, nel 1996 veniva distribuito per il grande pubblico, nel 2000 viene distribuito corredato da centinaia di programmi per ogni esigenza, siamo di fronte a un tasso di crescita con pochi eguali.

Torvalds ha intuitivamente trovato la via per guidare al successo la comunità di sviluppatori di cui era il referente guida.

Riassumendo la sua condotta possiamo dire che si è basata su alcuni capisaldi così descrivibili:

1. Ha messo a disposizione degli interessati un nucleo di codice funzionante, testabile e, soprattutto, estensibile senza limiti.

2. Ha organizzato la collaborazione in modo che ciascuno avesse la più ampia autonomia in creatività. Non esistevano ambiti ai quali gli sviluppatori erano rigidamente legati. Lo scambio delle informazioni era libero e incentivato. I membri più esperti fungevano da consiglieri e da primi revisori del codice. I membri meritevoli venivano inseriti nelle liste d'onore dei contribuenti.

3. Ha gestito il debugging impegnando al tempo stesso sviluppatori e utenti in un ciclo continuo di revisione e rilascio.

Riferendoci alle particolari caratteristiche della società hacker esposte da Eric Raymond in "Homesteading the Noosphere" (riportate in questa tesi

al Capitolo 2) possiamo riscontrare come le intuizioni di Torvalds si siano sposate con le aspettative del popolo della rete.

4. Torvalds ha fornito una noosfera (lo spazio delle idee e delle possibilità) pressoché illimitata. Presentando il nucleo di un piccolo sistema operativo esteso da funzionalità prese dal mondo Unix, presentò la possibilità concreta di ulteriori estensioni. Trattandosi di un sistema operativo poteva contare sulla combinazione di due fattori: un livello di complessità tale da mantenere vivo l'interesse e da non risultare insormontabile (il corretto equilibrio tra il senso della sfida e la possibilità di vincere) e l'infinita possibilità di implementare nuove possibilità (sempre nuovo spazio dove cimentersi).

5. Il progetto organizzato lasciando liberi i collaboratori di inserirsi nel settore preferito ha permesso l'utilizzo delle capacità personali e incontrato positivamente la mentalità libertaria degli hacker. Gli hacker, sentendosi liberi di agire come meglio credevano, hanno formato un gruppo compatto e ben affiatato. La relativa facilità di comunicazione in Internet ha sopperito alla mancanza di una forte direzione centralizzata.

6. Il rilascio continuo di versioni da provare ha funzionato come elemento regolatore dell'inevitabile caos che si può produrre in assenza di direttive precise. Ricordo che, secondo il modello di Raymond, il riconoscimento pubblico delle proprie capacità è la moneta con cui gli hacker ripagano il proprio lavoro. Portare le modifiche a pubblica revisione ha "obbligato" gli sviluppatori a impegnarsi in codice da cui poter ricevere stima e gli utenti, i primi "debugger", a segnalare le anomalie dopo averne approfondito la reale portata.

Quindi Torvalds sviluppò un progetto in pieno accordo con la mentalità generale dei suoi collaboratori.

Altri due elementi contribuirono al successo: l'entusiasmo per un sistema operativo alternativo e Internet.

Il successo del progetto fu dovuto anche all'ambizione del progetto stesso: creare non solo un sistema operativo, ma un sistema operativo alternativo ai sistemi operativi commerciali. L'odio anti Microsoft (per fare un nome) non può essere considerato il collante del gruppo di sviluppo di Linux, appartiene piuttosto a minoranze chiassose che alla comunità in generale. Ma se è vero

che non tutti gli hacker odiano i sistemi proprietari è anche vero che tutti ne farebbero volentieri a meno: la possibilità di partecipare alla creazione di un sistema operativo fu raccolta con entusiasmo.

Altro elemento fu il sapiente uso del mezzo di comunicazione: Internet. Server Ftp, mailing list, newsgroup furono massicciamente impiegati per mantenere lo scambio delle informazioni e per mantenere vivo l'interesse verso il progetto. Torvalds dimostrò un carisma eccezionale nel moderare i gruppi di discussione e promuovere le iniziative più interessanti. Sempre Torvalds garantì un equo riconoscimento dei meriti tra i collaboratori, guadagnandosi la fama di un giusto "gestore della fonte", per tornare al modello della società del dono.

Metaforicamente Torvalds mostrò la via verso nuove praterie ai pionieri della rete e mantenne unito il gruppo, senza bisogno di gerarchie e confini (del resto inapplicabili in un contesto come Internet), per quasi dieci anni.

È possibile sfruttare le intuizioni di Torvalds per un'economia di scala? La condizione necessaria perché il procedimento possa essere sfruttato è che possa venire riprodotto. Eric Raymond ci ha provato con Fetchmail, un programma che serve a scaricare la posta in locale da un server utilizzando il protocollo SMTP.

3.3 L'Emulo

Raymond, già menzionato più volte, è un attento studioso della comunità hacker e esponente di punta della Open Source Initiative. Uno dei suoi scritti più noti, "The Cathedral and the Bazaar", ha indirizzato la scelta della Netscape nel liberare il sorgente del loro prodotto Navigator.

Raymond, nel documento, analizza un progetto Open Source di successo da lui guidato: Fetchmail, utilizzato come test specifico per la verifica delle teorie sullo sviluppo del software suggerite dalla storia di Linux e mette a confronto due diversi stili di sviluppo: il modello "cattedrale" tipico del mondo commerciale e il modello "bazaar" del mondo Linux.

Raymond ha figurato le due metodologie con i termini "cattedrale" e "bazaar", ordinata e silenziosa la prima, aperta e rumorosa la seconda.

Il modello di sviluppo a "cattedrale" è tipico del mondo commerciale dove gli addetti al progetto lavorano gerarchicamente organizzati, ognuno con il proprio ruolo e funzione, dove le release vengono programmate in funzione

della clientela e dei tempi tecnici di sviluppo. Il debugging viene compiuto internamente al gruppo di progetto o viene affidato a unità specializzate, in ogni caso è un'operazione alla quale il cliente finale parteciperà solo suo malgrado causa la scoperta di nuovi errori. Le attività di sviluppo sono coperte da segreto, perlomeno nei dettagli ritenuti centrali, di conseguenza la comunicazione tra sviluppatori appartenenti a gruppi diversi è molto scarsa.

Il modello di sviluppo a "bazaar" è quello ideato da Torvalds (anche se esistono esempi di minore portata precedenti a Linux). Non esiste una gerarchia istituzionale, i ruoli sono quelli che ognuno riesce a ritagliarsi in un'economia di scambio puramente meritocratica e soprattutto, Raymond lo ritiene centrale, l'attività di correzione è supportata da utenti e programmatori. Il rilascio delle nuove versioni, più o meno stabili, serve ad allineare gli sviluppatori sul codice e a fornire l'ultima base disponibile per il testing. Il rilascio non è più quindi il risultato del lavoro ma diventa parte del lavoro.

Raymond, dopo aver analizzato il successo di Linux, decise di riprodurre il metodo di Torvalds per un progetto a cui si accingeva a lavorare.

Essendo il responsabile tecnico di un piccolo provider si era "abituato", come spiega lui stesso, a poter usufruire di un servizio di mail praticamente istantaneo. Volendo poterne usufruire anche da casa senza dover aprire una connessione remota con Telnet incominciò a pensare a un metodo per scaricare automaticamente la posta dal server del provider. Non partì da zero; cercando nelle collezioni di free software disponibili in Internet trovò dei programmi che rispondevano parzialmente a quello che cercava. Tali programmi usavano un semplice protocollo, il POP (Post Office Protocol), ma avevano qualche serio limite nella gestione della risposta. Raymond superò la limitazione estendendo il progetto originario. Decise poi di cambiare progetto di partenza quando si imbatté in Popclient, quest'ultimo consentiva l'utilizzo di più protocolli di posta e aveva un'impostazione più professionale. Ripartì da capo nello sviluppo.

Un fatto importante accadde quando si accorse che il titolare del programma su cui stava lavorando non aveva più interesse a proseguire nello sviluppo di nuove funzionalità. Si accordarono perché la gestione del codice passasse a Raymond. E questo è un'altro punto di forza dello sviluppo Open Source: quando chi gestisce un progetto decide di impegnarsi altrove passa il testimone a un volontario di provata competenza. Fu così che Raymond si ritrovò a gestire il progetto Popclient e, cosa ancora più importante, ne ereditò gli

utenti.

Torvalds aveva dimostrato come gli utenti, stimolati, potessero diventare la carta vincente di un progetto. Considerando che tra gli utenti vi sono anche hacker e che il codice sorgente è completamente disponibile è possibile che gli utenti diventino hacker veramente molto efficaci.

Raymond, avendo il presupposto base: un codice funzionante ma non completo, decise di seguire la strada di Linux. Il prossimo passo consisteva nel rilasciare molto velocemente le release. Occorse prima un buon lavoro di revisione del codice per renderlo facilmente leggibile e ben strutturato, in modo da agevolare le auspicabili estensioni. Raymond creò una mailing list per quanti volevano partecipare al progetto, inserì i nomi nella lista dei contributori di chiunque collaborava, annunciava in modo simpatico il rilascio delle nuove beta, plaudeva agli aggiustamenti e ai feedback. I risultati furono rapidi, il progetto partito a Settembre del 1996 raggiunse rapidamente quota 249 beta tester, nel Maggio 1997 raggiunse i trecento. Poi accadde una cosa inaspettata: parecchi chiesero di essere eliminati dalla lista perché ormai Fetchmail (nuovo nome dell'ex Popmail) funzionava molto bene. Probabilmente tale calo di utenti è parte del ciclo di vita di un progetto in stile bazaar.

Il vero punto di svolta per il progetto giunse quando un utente inviò un'estensione per scambiare la posta con il protocollo SMTP. L'uso del protocollo permetteva innovazioni e funzionalità molto snelle ma, d'altro canto, per ottenere un progetto ben strutturato occorreva un pesante lavoro. Raymond si trovò di fronte a un bivio: cambiare la struttura del progetto o procedere per patch? Optò per la prima soluzione riconoscendo la validità dell'idea del suo collaboratore. E questo fu un risultato del metodo bazaar: i collaboratori al progetto potevano avere idee così brillanti e innovative da richiedere il reengineering del progetto. Applicando una riforma così profondamente strutturale il nome venne mutato in “Fetchmail”.

Fetchmail è oggi un “application killer”, ossia uno di quei programmi che schiacciano i prodotti della concorrenza occupando tutta la nicchia di mercato utile. E, in questo caso, è Free Software!

3.4 Condizioni necessarie per l'avvio di un progetto in stile Bazaar

Lo stesso Raymond analizza e illustra le condizioni necessarie per l'avvio di un progetto bazaar.

È evidente come lo stile bazaar non consenta la scrittura del codice partendo da zero. Si possono fare test, trovare errori, migliorare il tutto, ma sarebbe molto difficile dar vita dall'inizio a un progetto in stile bazaar. La nascente comunità di sviluppatori deve avere qualcosa da far girare e da sperimentare. Quando s'inizia a costruire una comunità bisogna essere in grado di presentare una promessa plausibile. Non è detto che il programma debba funzionare particolarmente bene. Non deve mancare però di convincere i potenziali cosviluppatori che possa evolversi in qualcosa di veramente ben fatto nel prossimo futuro. Quando Linux e Fetchmail vennero diffusi pubblicamente, erano dotati di un design di base forte e attraente.

Non è necessario che chi guida il progetto abbia un elevato livello di intuizione e bravura, ma è assolutamente importante che sia capace di riconoscere le buone idee progettuali degli altri. Torvalds non è, o almeno non ancora, un genio del design inteso come innovazione, piuttosto ha dimostrato di possedere un buon istinto per la soluzione più sicura e più semplice. La vera qualità di Torvalds è il sapere riconoscere quando un'idea può funzionare e il saperla integrare nel kernel Linux.

Lo stesso Raymond riconosce che il successo di Fetchmail è dovuto all'idea di un suo collaboratore di usare il protocollo SMTP.

È chiaro che chi decide di guidare un progetto deve avere una buona abilità progettistica, ma soprattutto un buon senso dell'equilibrio tra innovazione e stabilità. L'innovazione è l'elemento fondamentale per la buona riuscita di un progetto ma può anche essere la causa del suo fallimento. Se Torvalds avesse deciso di implementare innovazioni fondamentali nel corso dello sviluppo di Linux ci sarebbero state poche probabilità che il sistema operativo alla fine risultasse stabile ed efficace come è ora.

Un altro elemento che Raymond ritiene molto importante per il coordinatore è la capacità di comunicare efficacemente con gli altri. L'interesse e la competenza tecnica non sono tutto per gestire una comunità, ci vuole anche carisma e, perché no, il sapersi accattivare la simpatia altrui.

3.5 Metodologie Open Source alternative

I casi Linux e Fetchmail non sono gli unici progetti ad aver avuto successo all'interno della comunità hacker e l'approccio di Torvalds non è l'unico possibile.

Prenderemo ad esempio il webserver più usato in Internet: Apache, e un linguaggio di scripting che ha contribuito alla diffusione web: Perl.

Apache è un webserver installato sul 53

Apache è il frutto di Apache Project, un'impresa collaborativa di sviluppo software con lo scopo di creare un Web server HTTP robusto, adatto alle soluzioni commerciali, ricco di caratteristiche e con codice sorgente disponibile.

Il progetto è gestito da un gruppo di volontari che usano Internet per comunicare, pianificare e sviluppare il server e le applicazioni correlate. Questi volontari sono conosciuti come Apache Group. In più, centinaia di utenti contribuiscono segnalando anomalie, inviando patch e esponendo idee nuove.

Apache Project vide la luce nel Febbraio del 1995 raccogliendo un HTTP daemon Free Software sviluppato all'Università dell'Illinois rimasto senza responsabile. Alcuni webmaster che utilizzavano quel demone HTTP decisero di collaborare nel lavoro di mantenimento e di sviluppo sul programma, lavoro sempre necessario a causa del grande sviluppo di tecnologia Web based. Il gruppo era costituito da otto programmatori.

Il primo compito fu l'organizzare tutte le patch e le estensioni che ciascuno aveva sviluppato per conto proprio, seguì un reenginering dell'intero progetto e già a Luglio poté essere distribuito una nuova versione del web server: Apache 0.8.8. Dopo un buon beta testing, il porting sulla maggior parte delle piattaforme in uso, il lavoro di documentazione, la creazione di possibili estensioni opzionali (i "moduli"), nel Dicembre 1995 venne rilasciato Apache 1.0.

Il processo di sviluppo di Apache è sostanzialmente diverso dal modello bazaar delineato da Raymond anche se conserva molte analogie.

Il gruppo di sviluppo di Apache nasce come gruppo "chiuso", sono esperti nella gestione di servizi web che hanno bisogno di uno strumento all'altezza delle loro necessità. Contrariamente al modello bazaar questi adottano un modello che si può definire "del buon dittatore". La comunità che ruota attorno ad Apache è divisa in due anelli concentrici: l'anello interno è costituito da quello che è il vero e proprio "Apache Group", o "core" come si definiscono,

l'anello esterno è costituito da utenti molto attivi e da collaboratori più o meno saltuari. I membri del gruppo centrale svolgono una funzione oligarchica: decidono per votazione quali soluzioni tecniche adottare e se ammettere altri all'interno del gruppo. Solo i membri del core possono accedere ai sorgenti originali, gli altri hanno a disposizione delle copie. È possibile entrare a far parte del gruppo se si collabora da almeno sei mesi e se si viene proposti da un membro del gruppo, il resto del gruppo vota per approvare la decisione. Nel caso vi siano nuove proposte tutti sono invitati a esprimere un parere, ma in caso di votazione faranno testo solo i voti dell'Apache Group.

Come si vede la gestione è diversa da quella adottata prima da Torvalds e poi da Raymond: l'Apache Group è retto da un "comitato" che ha il potere di decisione sulle attività dell'intero progetto e la comunità è gestita con un regolamento.

In analogia con il modello bazaar possiamo notare che anche Apache è partito da un codice relativamente stabile e bisognoso di estensioni, il mezzo di comunicazione è anche qui offerto da Internet e la collaborazione esterna viene incentivata (anche se all'interno di ben precise regole).

Un altro progetto di successo è PERL, un linguaggio di scripting per il web molto versatile e specializzato nel trattamento del testo. PERL è gestito da "The Perl Institute", un'associazione organizzata similmente ad Apache con la differenza che il bastone del comando ruota, a intervalli fissi di tempo, tra i membri del gruppo di gestione. Durante il proprio turno si è responsabili della gestione del progetto e competono i diritti di decisione sulle scelte da adottare.

In genere i nuclei al centro del progetto si occupano anche della raccolta di fondi per il sostentamento dell'attività di sviluppo (e degli sviluppatori!). L'attività di finanziamento avviene attraverso i più svariati canali: la consulenza, il merchandising, i contratti di assistenza, la raccolta di offerte. Un'altra forma di sovvenzionamento è rappresentata dal raccogliere per poi offrire proposte di lavoro per i collaboratori del progetto. Le forme di introito per i progetti Open Source verranno approfondite più avanti.

Quello che si vuole ancora sottolineare in questo paragrafo è che il modello Open Source non sottintende una specifica forma organizzativa ma piuttosto delinea un canone di sviluppo costituito dalla disponibilità del codice sorgente, dall'utilizzo delle release come strumenti di sviluppo e dal sapiente uso di Internet.

3.6 Economia Open Source

Finora si è discusso delle origini storiche e ideologiche del software Open Source e del metodo di sviluppo, ma si è deliberatamente evitato l'argomento economico. Eppure la stessa definizione "Open Source" è nata per rispondere a domande provenienti dal mondo commerciale. Domande sottintendenti la possibilità di creare business con software di cui non si possiede la proprietà nel senso classico del termine.

Eppure, è un dato di fatto, numerose società oggi investono e creano reddito basandosi su software di cui non hanno o di cui rendono liberamente fruibile la proprietà intellettuale. Il dato di fatto è rappresentato, tanto per citare dei casi, dalle società che distribuiscono Linux (Red Hat, SuSE, Caldera, Corel, Debian). Queste società non offrono una licenza d'uso poiché tale licenza è libera, offrono un servizio che consiste nel fornire su un supporto corredato da manuali software facilmente installabile e pronto all'uso, un servizio di assistenza e un servizio di aggiornamento delle patch. Ma queste rappresentano solo un caso dei tanti possibili. Più avanti si presenterà una casistica più esauriente.

È importante chiarire come il mercato del software sia profondamente legato al valore d'uso del software stesso.

Gli economisti distinguono tra valore d'uso e il valore di vendita di un bene. Il valore d'uso di un bene è il valore economico che possiede in quanto strumento, il valore di vendita è il valore economico a cui viene posto sul mercato per essere commerciato. Nel mondo dei produttori di software vi sono società che producono reddito sul valore d'uso del software, si pensi alle società di consulenza o alle società che forniscono servizi, e società che producono reddito sul valore commerciale del software, quelle che vendono pacchetti pronti all'uso. Le società che producono reddito sul software come valore commerciale sono molto poche, circa il 10-15

Un altro dato da considerare è che la maggior parte del software viene scritto all'interno di aziende che non sviluppano software come attività di reddito. È il caso del software gestionale (banche, assicurazioni, pubblica amministrazione, ecc.) e del software di controllo (dagli elettrodomestici agli airbus). Proporzionalmente il software scritto in loco alla struttura che lo richiede ha una quota superiore all'80-85

Questi dati dimostrano come l'attività preponderante per chi produce software è nel fornire un servizio ossia, commercialmente, nel stipulare un con-

tratto per determinati servizi, abbonamenti e scambio di valore continuativo tra produttore e consumatore. Ed è su questo campo, come abbiamo visto il più esteso, che il software libero sfida non soltanto metodologicamente ma anche economicamente il software tradizionalmente commerciale. Il software libero è rivolto per sua natura a offrire un servizio e non ad essere venduto in quanto tale. Considerato inoltre, come abbiamo visto, il rapido tasso di sviluppo di un progetto Open Source si può dire che il software libero sia un temibile concorrente per le tradizionali impalcature devolute ai servizi.

Il mercato del software basato sul valore di vendita ha dimostrato pesanti limiti derivanti dal fatto che utilizza il modello commerciale dei normali beni materiali. Il problema è che il software non può essere interamente assimilato a un qualsiasi altro bene per vari motivi: una volta creato non necessita dell'uso di particolari quantità di materie prime per essere replicato e distribuito (non è un'automobile), non ha un valore in sé come bene (non è oro), non acquista valore per scarsità (non è un'opera d'arte). Gli utenti ritengono accettabile il prezzo del software sulla base del servizio che questo è supposto rendere e sulla base dei servizi che fornirà il produttore. Questo è evidente nel caso una casa produttrice fallisca: i suoi prodotti non vengono più richiesti dal mercato. Il software è convenzionalmente trattato come un'industria manifatturiera benché tutto lasci intuire che sia un'industria di servizi. Il prezzo del pacchetto viene fissato a un livello solitamente molto alto ma tuttavia insufficiente a coprire i servizi di assistenza e manutenzione che in genere occupano molto più del 50

Questo modello porta a risultati deludenti per entrambe le parti: produttori e utenti. Gli utenti perdono perché in quel mercato non possono avere un servizio competente. I produttori ci rimettono perché avendo legato i loro introiti alla vendita di prodotti cercheranno di produrne sempre di nuovi e di risparmiare sul servizio di assistenza. Ma risparmiare sull'assistenza significa alienarsi clienti che presto o tardi passeranno ad altri produttori. L'unico modo di cavarsela in un mercato del genere è poter contare su un'espansione continua delle vendite, ipotesi poco realistica. Un'altra possibilità è conquistare tutto il mercato e detenere di fatto un monopolio, che in effetti è quello che è avvenuto nel mercato dei sistemi operativi per PC, per i word processor, per i fogli elettronici, per il software per l'impresa in generale. E comunque l'utente, anche in questo caso, perde.

L'alternativa è rappresentata dall'economia Open Source: vendere il ser-

vizio collegato al software. Perché può essere economicamente vantaggioso regalare il codice e guadagnare sui servizi ad esso collegati?

3.7 I vantaggi per l'utente

Supponiamo che una società stipendi programmatori per costruire un software di gestione della contabilità. Che vantaggio avrebbe a rendere liberamente disponibile il codice sorgente? A prima vista nessuno, anzi è probabile che i concorrenti utilizzino lo stesso software non dovendolo pagare e questo potrebbe trasformarsi in un vantaggio per la concorrenza. E in effetti se il software elaborato rappresentasse un reale vantaggio strategico allora non converrebbe certo fare in modo che tutti ne possano usufruire. Escluso il caso di software che costituisce un vantaggio strategico per la vita della società, che vantaggi si avrebbero a liberare il sorgente? Se è software valido altre società potrebbero decidere di utilizzarlo e molto probabilmente lo estenderebbero per coprire nuove o particolari funzionalità. Questo codice migliorato e/o esteso sarebbe disponibile, senza ulteriori spese, anche a chi ha sovvenzionato il progetto per primo. È vero che chi sovvenziona il progetto si sobbarca la spesa maggiore, ma avrebbe speso la stessa cifra anche se non avesse rilasciato il codice sorgente.

Un'ulteriore vantaggio sarebbe la garanzia di non perdere la knowledge base riguardante il programma. Se per un qualche motivo si perdesse la disponibilità dei programmatori che hanno sviluppato il codice non sarebbe un problema trovare altri programmatori che conoscono il codice e sarebbero in grado di intervenire per anomalie o estensioni funzionali. È in base a questo criterio che la Cisco decise di rendere Free Software un programma per la stampa di file in modalità remota. Il programma consentiva di stampare file su stampanti geograficamente lontane e se per qualche motivo la stampa non poteva essere effettuata la richiesta sarebbe stata dirottata sulla stampante più vicina alla destinazione. I due autori del programma si resero conto che nel caso avessero lasciato la Cisco nessuno sarebbe stato più in grado di gestire il programma e questo sarebbe diventato presto obsoleto o non in grado di gestire eventuali modifiche ambientali. Proposero allora di poter rilasciare il codice in modo che il programma avrebbe potuto essere adottato da altre società, garantendo così la diffusione della conoscenza del programma. La Cisco seguì il consiglio considerando che poteva ricavarne solo vantaggi: il

programma avrebbe continuato ad essere tecnologicamente aggiornato anche senza la stretta collaborazione degli autori e in caso fossero state necessarie modifiche si sarebbe sempre potuto trovare personale già a conoscenza del software.

Un altro vantaggio per l'utente consiste nel fatto che, utilizzando software Open Source, si è maggiormente liberi nella scelta del fornitore. Infatti l'Open Source consente di sganciarsi da un fornitore senza dover riacquistare l'intero parco software. Il codice liberamente disponibile agevola l'uso e la diffusione di standard aperti, cioè liberamente utilizzabili, di conseguenza è possibile cambiare fornitore o accedere a nuovi servizi senza dover cambiare il software che già si possiede. Nel caso fosse indispensabile invece sostituire il software utilizzato almeno le spese di acquisto delle nuove licenze sarebbero nulle o molto basse. Questo consentirebbe un più accessibile uso di nuove tecnologie.

3.8 I vantaggi per il programmatore

I programmatori che collaborano a progetti Open Source in genere hanno le stesse necessità dei programmatori che lavorano inseriti in un contesto commerciale. Queste necessità comprendono il nutrirsi, il vestirsi, abitare in una casa, mantenere la famiglia e garantirsi una sufficiente prosperità. Tutto ciò nel mondo commerciale è garantito, o dovrebbe, dalla retribuzione contrattuale o dalla vendita di un bene, in questo caso un programma. Nel mondo del Free Software il programma, come abbiamo visto, non costituisce di per sé un valore. Perché allora potrebbe convenire per un programmatore distribuire gratuitamente un sorgente? Affrontiamo un caso esemplificativo. Supponiamo che un programmatore costruisca una patch di una certa rilevanza, avrebbe di fronte tre possibilità: tenersela, venderla, distribuirla con una licenza Open Source. Valutiamo i casi. Nel primo, non distribuire il proprio lavoro, non avrebbe palesemente nessun vantaggio economico. Nel secondo caso, vendere il codice binario, il programmatore andrebbe incontro a qualche problema di natura logistica. Ad esempio: come farsi pagare? Attualmente i mezzi di scambio monetario su Internet non sono molto convenienti per piccoli volumi di vendita, altri mezzi potrebbero allontanare possibili clienti (es.: i vaglia postali) o non essere disponibili all'intera clientela (es.: pagamenti internazionali). Anche supposto che venga messo a punto un sistema di vendita sicuro, rapido e accessibile si dovrebbe poi decidere quanto far pagare la patch. Il

prezzo di vendita di un programma è ritenuto accettabile dagli acquirenti in base ai vantaggi, al “servizio”, che il programma stesso fornirà. Se il programmatore sviluppa patch a tempo perso allora la valutazione del prezzo ha poca importanza, questo influirà solo sul numero di clienti. Ma se sviluppare patch è l’attività principale il discorso cambia perché si ricade nella spirale del software commerciale delineata prima. Ossia è molto probabile che i costi di manutenzione progressivamente si mangeranno i guadagni derivati dalle vendite, a meno che l’espansione del mercato non sia tale da garantire sempre nuovi introiti. E questa è una scommessa molto rischiosa.

Il terzo caso è rappresentato dalla distribuzione Open Source. Non si ha nessun guadagno nel distribuire il proprio codice ma si ha la possibilità che qualcuno, riconoscendo nel software distribuito delle capacità, richieda la propria competenza, e sia disposto a pagarla, per altri progetti.

Questa modalità è in particolare utilizzata dalle organizzazioni che patrocinano progetti Open Source come il “Perl Institute” o il “PHP Core”. Queste organizzazioni, tra le altre cose, raccolgono fondi per i progetti che guidano. La risorsa principale di queste organizzazioni sono i programmatori che collaborano, non restribuiti, al progetto. Un modo per consentire a più programmatori possibile di avvicinare il progetto consiste nell’offrire possibilità di lavoro retribuito da parte di committenti esterni. Ad esempio il PHP è un linguaggio di scripting molto efficiente, il PHP Core si pone da tramite tra chi ha bisogno di commissionare la creazione di siti web o di particolari servizi e i programmatori legati al progetto PHP, che naturalmente conoscono molto bene la problematica. In questo modo il PHP Core ottiene molteplici risultati: catalizza l’attenzione dei committenti sul proprio prodotto, attira nuovi programmatori, coinvolge i programmatori nel progetto offrendo la possibilità di guadagnare lavorando su ciò che già facevano gratis.

Inoltre il sorgente libero, essendo gratuito, permette ai programmatori di tenersi al corrente delle nuove tecnologie o comunque di non fossilizzarsi nel ristretto ambito del proprio lavoro e quindi di potersi garantire una “mobilità” mirata.

Vi è perlomeno ancora un caso in cui ai programmatori conviene l’uso di software Open Source. È il caso, già citato, di Apache Project. I programmatori che si unirono per primi avevano l’esigenza di poter usare un web server sufficientemente adatto alle proprie esigenze. Non trovando in commercio nulla di soddisfacente decisero di unire gli sforzi nella costruzione del

proprio web server utilizzando e condividendo codice libero, decisero poi di raccogliere i contributi di ulteriori volontari. In questo caso il codice libero ha permesso la creazione dello strumento che permettere di risolvere le esigenze economiche di quei signori (e, proprio nel caso di Apache, di molti altri che lo impiegano sul proprio Web Server).

3.9 I vantaggi per l'impresa fornitrice

Raymond, in "The Magic Cauldron", analizza il fenomeno Open Source dal punto di vista economico. Nel suo documento individua alcuni modelli di riferimento per le imprese che hanno il software come attività di reddito. Bisogna considerare che l'Open Source è un modello molto "giovane", di conseguenza le dinamiche economiche che l'accompagnano sono in piena evoluzione. Fondamentalmente l'Open Source permette l'apertura di mercati nei servizi legati al software che comportino un valore di vendita indiretto. Per la precisione Raymond espone cinque modelli commerciali utilizzati e due modelli teorici.

3.9.1 Articolo civetta/posizionatore sul mercato

In questo modello, si usa il software open source per creare o mantenere una posizione sul mercato per un software proprietario che genera una fonte diretta di profitti. Nella variante più comune, un software client "libero" agevola le vendite di un software server, o profitti da abbonamento/pubblicità in relazione a un portale Web.

La Netscape Communications perseguiva questa strategia quando, all'inizio del 1998, rese disponibile il codice sorgente del browser Mozilla. Il giro d'affari legato al browser ammontava al 13

In effetti, rendendo disponibile il codice sorgente del browser Netscape, ancora assai diffuso, Netscape ha negato alla Microsoft la possibilità di istituire un monopolio del browser. Netscape prevedeva che lo spirito di collaborazione dell'open source avrebbe accelerato lo sviluppo e il debugging del browser. Sperava altresì che Microsoft Internet Explorer si sarebbe trovato costretto a tenere il passo e dunque impossibilitato a definire l'HTML in modo esclusivo.

Questa strategia ha funzionato. Nel novembre 1998, infatti, Netscape ha cominciato a riguadagnare la sua quota di mercato rispetto a Internet Explorer. Al momento dell'acquisizione di Netscape da parte di America On-Line (AOL), all'inizio del 1999, il vantaggio competitivo di aver mantenuto in gioco Mozilla era abbastanza chiaro perché uno dei primi impegni dichiarati da AOL è stato proprio quello di continuare a sostenere il progetto Mozilla, benché si trovasse ancora allo stadio iniziale.

3.9.2 "Widget frosting"

Questo modello riguarda i produttori di hardware (hardware, in questo contesto, comprende tutto a cominciare da schede Ethernet e simili fino a interi sistemi). Le pressioni del mercato hanno costretto le società che trattano hardware a scrivere e gestire software (dai device driver agli strumenti di configurazione, fino a interi sistemi operativi), ma il software stesso non è fonte di profitti, bensì una spesa aggiuntiva, spesso consistente.

In questa situazione, la disponibilità del codice sorgente non dà molto da pensare. Non ci sono guadagni da perdere, né aspetti negativi. Ciò che il produttore acquista è una risorsa di sviluppo di gran lunga maggiore, una risposta più rapida e flessibile alle esigenze dei clienti e maggiore affidabilità, grazie alla possibilità di revisione reciproca. Inoltre, apre la strada ad altri ambienti gratuitamente e, probabilmente, accresce anche la fedeltà dei clienti, in quanto il personale tecnico dedica più tempo al codice per esaudire le richieste di personalizzazione della clientela.

Il margine di sicurezza dell'open source è particolarmente evidente nel campo del debugging di elementi di interfaccia ("widget frosting"). I prodotti hardware hanno produzione e tempi di assistenza limitati, scaduti i quali, i clienti si ritrovano da soli. Ma se i clienti avessero accesso al codice dei driver, con la possibilità di utilizzarli per produrre patches secondo il bisogno, è più probabile che rimangano clienti della stessa società.

Un esempio assai illuminante dell'adozione del "widget frosting" ci è fornito dalla decisione da parte di Apple Computer, a metà marzo del 1999, di rendere disponibile il codice sorgente di "Darwin", il fulcro del sistema operativo MacOSX server.

3.9.3 Rivelare la ricetta, aprire un ristorante

In questo modello, si rende disponibile il codice sorgente di un software per creare una nicchia di mercato non nel settore del software commerciale (come nel caso Articolo civetta/posizionatore sul mercato), bensì nei servizi.

Ecco come agiscono Red Hat e altri distributori di Linux. Ciò che vendono, in realtà, non sono i software, ossia i bit, ma il valore aggiunto nell'assemblaggio e nelle prove di un sistema operativo funzionante e con garanzie (anche solo implicite) di commerciabilità e compatibilità con altri programmi portanti lo stesso marchio. Altri elementi del valore da essi offerto comprendono l'assistenza gratuita per l'installazione e l'offerta di opzioni per contratti di assistenza continuativi.

Le possibilità di costruire mercati nell'open source sono estremamente significative, specie per imprese che si trovino, per propria natura, nel settore dei servizi. Un caso recente e assai istruttivo è rappresentato da Digital Creations, un'impresa di Web design aperta nel 1998 e specializzata in banche dati complesse e siti per la gestione di transazioni. Il suo strumento principale, la punta di diamante della proprietà intellettuale dell'impresa, è un object publisher che oggi, dopo essere passato attraverso diversi nomi e forme, si chiama Zope.

Quando Digital Creations decise di cercare nuovi investitori, l'esperto convocato valutò la futura nicchia di mercato dell'impresa, il suo personale e i suoi strumenti, per poi consigliare alla Digital Creations di portare Zope all'open source.

Stando agli standard tradizionali dell'industria del software, questa sembrerebbe una mossa completamente folle. La saggezza convenzionale, acquisita nelle scuole di economia, ci insegna che una proprietà intellettuale vitale come Zope rappresenta la punta di diamante di un'impresa e che non deve essere venduta in alcun caso. Ma l'esperto aveva due argomenti a suo sostegno: il primo, che il vero patrimonio di Zope, in realtà, erano i cervelli e le abilità delle persone in esso impegnate; il secondo, che probabilmente, Zope avrebbe prodotto più valore nel costruire un mercato che come strumento segreto.

Per capire tutto questo, paragonate le due situazioni possibili. In quella convenzionale, Zope rimane l'arma segreta della Digital Creations. Ammettiamo pure che sia molto efficace. Il risultato è che l'impresa sarà in grado di ottenere una qualità superiore in tempi brevi, ma nessuno lo saprà. Sarà

facile soddisfare i clienti, ma più difficile sarà iniziare con l'accattivarsi una clientela di base.

L'esperto, invece, capì che uno Zope open source avrebbe rappresentato una pubblicità importantissima per il vero patrimonio della Digital Creations: le persone. Ipotizzò che i clienti, valutando le prestazioni di Zope, avrebbero considerato più efficace contattare gli esperti, piuttosto che sviluppare una gestione interna di Zope.

Da allora, uno dei rappresentanti di Zope conferma pubblicamente che la strategia open source ha "aperto molte porte che non si sarebbero trovate altrimenti". I potenziali clienti non mancano di rispondere alla logica sottesa a questa situazione e, come previsto, Digital Creations è in auge.

Un altro esempio recentissimo è fornito da "e-smith, inc". Questa società vende contratti di assistenza per un server Internet pronto all'uso e open source, un adattamento di Linux. Uno dei rappresentanti, descrivendo la diffusione dei software e-smith scaricati gratuitamente, ha affermato: "Per la maggior parte delle imprese questa sarebbe pirateria del software: per noi è libero mercato".

3.9.4 Fornire accessori

In questo modello, si vendono accessori per il software open source. Allo scalino più basso, tazze da tè e magliette; a quello più alto, documentazione redatta e prodotta a livello professionale.

La O'Reilly Associates, editrice di molti manuali eccellenti sul software open source, è un ottimo esempio di impresa fornitrice di accessori. Per la precisione, O'Reilly impiega famosi hacker operanti nell'open source (tra cui Larry Wall e Brian Behlendorf) come redattori di manuali, come metodo per costruirsi una reputazione nel mercato di sua scelta.

3.9.5 Liberare il futuro, vendere il presente

In questo modello, si rilascia il software in file binario e codice sorgente con una licenza commerciale, ma contenente una data di scadenza della non disponibilità del codice sorgente. Ad esempio, si potrebbe scrivere una licenza che permetta la libera ridistribuzione, proibisca l'uso commerciale senza paga-

mento di contributi e garantisca che il software rientri nei termini della licenza GPL un anno dopo il rilascio o in caso di cessata attività del produttore.

Con questo modello, i clienti hanno la certezza che il prodotto sia personalizzabile secondo i bisogni, in quanto ne hanno il codice sorgente. Il prodotto è ad alto margine di sicurezza: la licenza garantisce che una comunità open source possa rilevare il prodotto, qualora l'impresa originale si estingua.

Poiché il prezzo e il volume delle vendite si basano su queste aspettative del cliente, l'impresa originale dovrebbe vedere aumentare i propri profitti grazie a un prodotto di questo tipo, più che con una licenza esclusivamente commerciale. Inoltre, dal momento che il codice originale è sotto licenza GPL, sarà comunque passato in rassegna seriamente da più persone, subirà correzioni e altri aggiustamenti che solleveranno l'autore da una parte del 75

Questo modello è stato seguito con successo da Aladdin Enterprises, gli autori del famoso programma Ghostscript (un interprete PostScript in grado di tradurre nel linguaggio originale di molte stampanti).

Lo svantaggio principale di questo modello è che le clausole commerciali tendono a inibire la revisione reciproca, nonché la partecipazione, proprio agli inizi del ciclo di produzione, quando ce n'è più bisogno.

3.9.6 Liberare il software, vendere il marchio

Questo modello rappresenta una possibile strategia aziendale. Si procede all'open sourcing di una tecnologia software, mantenendo una sequenza di prove o insieme di criteri di compatibilità, per poi vendere agli utenti un marchio che certifichi la compatibilità dello strumento tecnologico in loro possesso con tutti gli altri della stessa marca. (È così che Sun Microsystems dovrebbe gestire Java e Jini.)

3.9.7 Liberare il software, vendere il contenuto

Questo modello, ancora una volta, rappresenta una possibile strategia aziendale. Immaginiamo una specie di servizio di borsa telematica in abbonamento. Il valore non è da ricercare né nel software client, né nel server, bensì nell'oggettiva affidabilità delle informazioni fornite. Quindi, si rende disponibile il codice sorgente di tutti i software e si vendono abbonamenti per accedere

ai contenuti. Via via che gli hacker aprono il client a nuove prestazioni e lo migliorano in vari modi, il mercato si espande automaticamente.

(Ecco perché AOL dovrebbe effettuare l'open sourcing del suo software client.)

3.10 I criteri per scegliere la strada Open Source

L'Open Source non è la panacea di tutti i mali e bisogna tenere presente che la maggior parte dei progetti software, sia commerciali che Open Source, non raggiunge un buon esito. È comunque possibile individuare dei criteri, ed è ancora Raymond che si dimostra l'osservatore più attento del fenomeno, con cui poter decidere quando effettivamente è conveniente percorrere la via Open Source.

Come abbiamo visto fino qui l'Open Source consente di raggiungere risultati di qualità, stabilità e misurabilità in tempi relativamente brevi, inoltre consente di risolvere efficacemente, attraverso la autonoma revisione reciproca, problemi connessi alla correttezza del design e dell'implementazione. Quando questi fattori sono inerenti lo sviluppo di un software è bene tenere in considerazione lo sviluppo a sorgenti liberi.

Il software Open Source promuove di fatto standard aperti, di conseguenza permette una maggiore indipendenza dal fornitore. Sia che il fornitore non sia più in grado di svolgere la propria funzione o per scelta sarà più facile, e meno dispendioso, trovare un sostituto. Questa possibilità è particolarmente sentita quando il software diventa "centrale" per la vita di un'impresa.

Nel settore delle applicazioni l'Open Source raggiunge il massimo dell'efficacia nell'aumentare i profitti rispetto al software commerciale per software che istituiscano o abilitino una comune infrastruttura di calcolo e comunicazioni, questa caratteristica è profondamente legata alla facile espansione sul mercato di software libero. Per le società che usano il software per vendere un servizio (si pensi all'intermediazione finanziaria via Internet per piccoli investitori) converrebbe utilizzare il software Open Source come veicolo d'espansione del proprio mercato. Il software libero si diffonde molto velocemente allargando così la base dei possibili clienti, inoltre, con la possibilità d'uso del codice sorgente sarebbe possibile approfittare di un naturale processo di innovazione portato avanti dagli stessi utenti.

Un altro fattore da prendere in considerazione è il livello tecnologico del

software: se si basa su conoscenze o algoritmi base largamente noti ha poco senso temere un danno dalla diffusione del proprio know-how. Ben diverso il caso di fornitori di servizi unici o altamente differenziati, questi hanno più ragione di temere la copia dei loro metodi da parte dei proprietari.

Il core software di Internet, Apache e l'implementazione Linux dell'Application Program Interface Unix con standard ANSI rappresentano esempi canonici dei criteri esposti.

Riassumendo, le seguenti discriminanti fanno propendere per l'Open Source:

1. quando affidabilità, stabilità, misurabilità sono fondamentali
2. quando la correttezza del design e dell'implementazione non possono essere verificate efficacemente se non tramite revisioni reciproche e indipendenti
3. quando il software è di vitale importanza per il controllo dell'impresa da parte dell'utente
4. quando il software abilita una comune infrastruttura di calcolo e comunicazioni
5. quando i metodi di punta (o loro equivalenti funzionali) fanno parte delle comuni conoscenze ingegneristiche

Bisogna anche valutare che la rispondenza ai criteri presentati da parte di un software può mutare nel tempo. È il caso ad esempio dei giochi, in particolare di Doom.

Doom uscì sul mercato a fine 1993 e presentava, dal punto di vista grafico, innovazioni tali da schiacciare la concorrenza. Le caratteristiche stesse del programma ne sconsigliavano il rilascio come software Open Source: il vantaggio tecnologico, non era difficile da porre a verifica, non avrebbe tratto benefici da una diffusione gratuita e comportava costi tollerabili in caso di guasto. E in effetti fu un grosso successo commerciale. Col tempo la concorrenza recuperò lo svantaggio guadagnando sempre più quote di mercato, nel frattempo i programmatori di Doom erano impegnati nelle estensioni del gioco (multi-user, multi-piattaforma, scontri diretti in rete) invece che, come avrebbero preferito, dedicarsi al prossimo gioco. I bit segreti di Doom

perdevano sempre più valore man mano che uscivano nuovi giochi, fu così effettuata una scelta di mercato: venne rilasciato il codice sorgente e si decise di impegnarsi commercialmente nelle antologie di scenari. I programmatori furono impegnati nello sviluppo di nuovi giochi e Doom libero consentì alla società proprietaria di cavalcare l’onda degli scenari.

Di conseguenza la scelta tra software libero e il software commerciale dipende anche dalla situazione contingente di mercato: ciò che oggi potrebbe convenire come commerciale domani potrebbe convenire come Open Source.

3.11 Open Source all’interno dell’azienda

L’Open Source potrebbe venire impiegato all’interno delle grosse aziende per sopperire alla difficoltà di condivisione dell’informazione. Quando una società si avvale di più gruppi di sviluppo impegnati su più progetti in genere non vige un reale scambio delle informazioni tra gli addetti ai lavori. Tale scarsità di comunicazione si rivale sul processo produttivo aumentando i costi e di conseguenza diminuendo la competitività. I costi aumentano perché aumenta la quantità di lavoro ridondante e quindi inutile, aumenta la richiesta di consulenze esterne per problemi già risolti all’interno dell’azienda, aumentano i tempi di progetto perché non si è incentivata una reale cultura del riutilizzo. Il problema è sentito e pubblicamente riconosciuto tuttavia le soluzioni proposte mancano il bersaglio creando così ulteriori problemi.

Quando le grandi società decidono di ottimizzare il processo di sviluppo del software in genere sono consigliate, da altrettanto grandi società di consulenza, di adottare un modello standard di progettazione, di adottare regole comuni di descrizione del codice per tutti gli sviluppatori, di organizzare periodiche riunioni tra gli addetti ai lavori per lo scambio di esperienze. Saggi consigli che in pratica non vengono utilizzati perché le riunioni vengono considerate, diventando presto ripetitive, perdite di tempo, gli standard richiedono tempi non indifferenti di apprendimento, i progetti evolvono e controevolvono in tempi molto più rapidi della documentazione che dovrebbe accompagnarli e, costante assoluta nel mondo commerciale: se il tempo a disposizione è scarso per lo sviluppo, figurarsi il tempo a disposizione per la documentazione! I consigli classici sull’organizzazione dello sviluppo software falliscono perché non sono adatti alle realtà produttive, di più: cercano di adattare l’attività produttiva a canoni certamente corretti ma non compatibili con le esigenze di

mercato. Tutti sono pronti a riconoscere l'importanza degli standard di documentazione a qualsiasi livello ma quando ci si accorge che la documentazione non crea reddito, almeno nel breve e medio periodo, per l'azienda e che non crea riconoscimento per l'autore i buoni propositi vengono schiacciati dalle situazioni contingenti. Ogni società economica persegue il profitto cercando di abbassare i costi e di aumentare le entrate. Per le società che producono informatica il tempo di sviluppo è il costo maggiore e la documentazione costa tempo, è quindi chiaro che le società stesse siano più propense ad utilizzare le proprie risorse per affrontare nuovi progetti piuttosto che a documentare utilmente quelli già in corso. Tuttavia rimane di fondamentale importanza lo scambio delle conoscenze tra i gruppi di lavoro e riuscire a gestire bene il "know how" interno è un fattore strategico.

Per guadagnare tale vantaggio bisogna percorrere la strada inversa a quella da sempre prospettata: invece di adattare l'azienda ai sacri canoni dello sviluppo software bisogna adattare i sacri canoni all'azienda. Alcune peculiarità dello sviluppo Open Source possono risolvere la questione efficacemente. Il nocciolo è inserire nella cattedrale un po' del lievito del bazaar, senza bisogno di sovvertire gerarchie. Esporrò le linee guida.

1. Lo scopo di documentazione e standard è rendere disponibile la conoscenza in oggetto nel modo più pratico possibile. Il fatto di renderla disponibile e accessibile a tutti ne agevola molto il reperimento. Quindi dovrebbe essere tenuta, anche quella ancora in fase di sviluppo, su un file server accessibile da tutti i dipendenti. Non occorre che tale documentazione sia particolarmente esauriente, l'importante è che per ogni progetto sia facilmente accessibile un registro contenente la problematica risolta, gli strumenti adottati, i problemi incontrati, le soluzioni trovate e soprattutto informazioni riguardo la reperibilità di chi si è occupato del progetto.

2. Unitamente al progetto dovrebbe essere disponibile il codice attinente. Questo permetterebbe di raggiungere un duplice risultato: agevolare il riutilizzo dei componenti e invogliare i tecnici a scrivere codice leggibile e corredato di adeguata documentazione. Come nei progetti Open Source si instaurerebbe un circolo virtuoso: leggendo la documentazione e il codice di altri si è spronati a produrne di proprio all'altezza delle aspettative.

3. Agevolare il contatto mirato tra gli addetti ai lavori. Le riunioni periodiche servono a poco se non c'è un preciso interesse nei partecipanti. È più produttivo ed elastico l'uso di strumenti come le mailing list e i newsgroup: quando qualcuno ha un problema può richiedere l'attenzione dei colleghi via mail. In questo modo è anche agevolata la creazione di un archivio che tenga traccia dei problemi risolti e la redazione di documentazione FAQ (Frequently Asked Questions). L'uso di questo tipo di documentazione è spesso sottovalutato in ambito commerciale, ma sono risorse di primaria importanza soprattutto per i neo assunti che accelererebbero l'apprendimento diventando produttivi quanto prima.

Potrebbero sorgere alcune obiezioni, ad esempio riguardo la sicurezza. Il know-how è il patrimonio delle aziende informatiche, concentrare la documentazione in un solo posto liberamente accessibile dai dipendenti può facilitare dolose "fughe" di notizie. Un'altra obiezione potrebbe essere legata al fatto che mailing list e newsgroup richiedono manutenzione, di conseguenza il tempo risparmiato in riunioni e nel reperimento delle informazioni verrebbe impiegato nella gestione dell'infrastruttura.

Riguardo la prima obiezione si può rispondere che: a) è da valutare il fatto che la documentazione sparsa per gli uffici sia un dato di sicurezza, b) non è strettamente necessario che tutto sia a disposizione di tutti, l'importante è che tutti possano almeno sapere a chi rivolgersi per un problema che sanno già essere stato risolto.

L'aumento del lavoro per mantenere l'infrastruttura per lo scambio di informazioni è innegabile, ma anche ammesso che sia pari al tempo risparmiato nello sviluppo (e non lo è) resta il fatto che si crea valore per l'azienda. In caso di cessione o di vendita di alcuni progetti vi sarebbe un innegabile valore aggiuntivo in documentazione.

Il libero uso della documentazione permetterebbe inoltre un agevole uso del personale che potrebbe più facilmente essere trasferito da un progetto all'altro.

4 Un caso reale: il progetto “SEPRA”

Ho avuto modo di verificare la potenza e la versatilità di alcuni strumenti Open Source nel corso della mia esperienza come libero professionista. Il progetto più rappresentativo è stato il progetto “S.E.P.R.A.”, o SEPRA d’ora in avanti. SEPRA è la sigla di “Sorveglianza Epidemiologica Patogeni Respiratori Ambulatoriali”, un progetto per la sorveglianza di alcuni batteri patogeni responsabili delle infezioni all’apparato respiratorio. La sorveglianza in questo caso significa il monitoraggio della resistenza ad alcuni tipi di antibiotici dei microrganismi oggetto di osservazione. Il monitoraggio viene eseguito analizzando campioni prelevati da pazienti infetti ricoverati in strutture sanitarie in dieci località italiane, il test viene eseguito semestralmente e mediante opportuni misuratori si è in grado di rilevare il livello di resistenza agli antibiotici.

Il SEPRA ha il duplice di scopo di fornire dati scientifici per la ricerca medica e di diffondere un nuovo strumento di analisi fornito dalla Bayer. Lo studio viene eseguito in dieci strutture ospedaliere ed è coordinato dalla CHA (Milano, rif. Sig. Lino Braceschi), società leader in Italia per il promoting di strumenti medici, responsabile medico del progetto è il Dott. Giancarlo Schito (Genova, Ospedale S. Martino). Essendo la ricerca dislocata in centri geograficamente non vicini la CHA ha valutato l’idea di raccogliere i dati via Internet, mediante un sito che consentisse l’inserimento dei dati in pagine web appositamente preparate. Il sito ha compiti di raccolta, ordinamento e di una prima elaborazione dei dati. I dati raccolti vengono poi trasferiti alla società Biomedical (Padova, rif. Dott.sa Zambrotta) per una successiva elaborazione a fini scientifici. La CHA ha affidato la realizzazione del sito per la raccolta dati alla società Webline (Valenza Po, rif. Dott. Luca Pagella) che a sua volta ha chiesto la collaborazione del sottoscritto per la progettazione e realizzazione del servizio.

Ammetto che è stata la mia prima esperienza con strumenti Open Source, per questo decisamente molto significativa!

Per ovvie ragioni di riservatezza professionale non riporterò dati riguardanti la ricerca medica, sono comunque convinto che il senso del lavoro svolto non dipenda dai dati specifici del caso.

4.1 Le specifiche

Il lavoro di raccolta delle specifiche del progetto non è stato semplice a causa del fatto che nessuno dei committenti principali aveva esperienza con l'impostazione di un progetto informatico. Si è così proceduto per interviste e presentando delle bozze delle pagine che avrebbero raccolto e presentato i dati. Le specifiche del progetto possono essere così riassunte:

1. Il SEPRA è un servizio di raccolta dati strettamente riservati. L'utente responsabile della ricerca può visionare e modificare i dati di tutti. Gli utenti ricercatori possono inserire e visionare solo i propri dati, ma non possono modificarli successivamente. L'utente gestore (CHA) e l'utente di elaborazione finale (Biomedical) possono visualizzare i dati di tutti. I dati sono da ritenersi coperti da segreto industriale e pertanto è opportuno risiedano su una banca dati protetta.

2. L'interfaccia del sito dedicato alla raccolta dati deve essere progettata in funzione di un'utenza attenta e preparata, ma scarsamente pratica con i mezzi informatici

3. Ogni test va inserito su una scheda relativa ad un microrganismo riportante i dati del paziente infetto e comprende un numero fisso di antibiotici da testare

4. I microrganismi hanno valori di reazione per ogni antibiotico su scale diverse non compatibili tra di loro, inoltre sono oggetto di test specifici. Bisogna considerare il caso di test non riusciti e quindi riportanti valori fuori dalle scale assegnate.

5. L'utente di elaborazione finale dei dati deve poter scaricare i dati dal web server in modo semplice e in qualsiasi momento

6. La modalità di raccolta dati (via Internet) è da considerarsi come progetto pilota, per cui sono assegnati fondi limitati

7. Il numero di schede inserite a fine ricerca è da stimarsi sulle cinquemila unità

Dalle specifiche si è desunto che:

1. L'interfaccia deve necessariamente essere personalizzata per ogni tipologia di utente connesso

2. L'interfaccia deve essere di semplice utilizzo ma assolutamente precisa nei valori e nelle possibilità di interazione

3. L'utente deve poter riconoscere la scheda di test come fosse cartacea (cioè del tipo alla quale è abituato)

4. Il database deve contenere i dati relativi al test per ogni coppia microrganismo/antibiotico, inoltre deve contenere i test specifici per ogni singolo microrganismo.

5. È necessario progettare un estrattore dati e un modulo per la composizione del file richiesto, oltre a un metodo di invio semplice e sicuro

6. È necessario limitare l'acquisto di licenze per aumentare il profitto, allo stesso tempo occorrono mezzi di sviluppo sicuri e collaudati

7. La mole dati non rappresenta un fattore vincolante

I punti 1) e 4) obbligano alla creazione di pagine in HTML dinamico, ossia pagine che modificano la propria struttura dipendentemente dal tipo di utente e dai dati sul database. Il punto 6) ha spinto la scelta sul famigerato (per allora) software Open Source.

4.2 La scelta degli strumenti

Il web server è stato fornito dalla Webline affittando un server virtuale con sistema operativo Free BSD (Unix compatibile). Nelle condizioni di contratto è prevista una buona personalizzazione del server che è stato dotato di Apache 1.2 come web server. Per quanto riguarda il database e il linguaggio di scripting, dopo un'attenta consultazione con colleghi più esperti nel campo, si è deciso di optare su database PostGreSql e linguaggio di scripting PHP 3.12.

Il database Postgresql è un prodotto Open Source liberamente disponibile, tecnicamente è un db relazionale ad oggetti, fornisce cioè una sintassi che permette di dichiarare oggetti e relazioni tra di essi, in seguito genera le tabelle vincolate dalle relazioni dichiarate. Il prodotto si è presentato molto versatile e innovativo, ha presentato comunque alcune limitazioni per quanto riguarda le possibilità fornite dal motore di ricerca: infatti all'epoca (Novembre 1999) non supportava l'outer join. Postgresql era un prodotto giovane e la lista dei TODO era ancora lunga. Optai quindi per il meno ricco ma più affidabile MySql, un altro database Open Source di cui esiste una ricca documentazione. Il PHP 3 si è invece dimostrato la scelta corretta fin dall'inizio: semplice, potente, progettato per l'integrazione in pagine HTML e altamente versatile nelle connessioni al database, offre centinaia di funzioni che coprono ogni necessità relativa al parsing e alla generazione di pagine HTML dinamiche. Oltretutto in Internet esistono vaste banche di esempi e di codice free pronto ad essere riutilizzato.

Per poter lavorare senza la necessità di un perenne collegamento in remoto decisi di ricreare sul mio PC il sito e la dotazione software di stanza sul server remoto. Poiché Free BSD non era indicato per l'installazione su un PC domestico mi orientai verso Linux, la versione distribuita da Red Hat (Red Hat 6.1). L'installazione fu relativamente semplice, l'interfaccia utente guida ogni passo e le modifiche a disposizione dell'utente sono davvero molto limitate. Più che altro il sistema interroga l'utente su cosa si legge dal monitor per avere conferma dei driver installati. Terminata l'installazione incominciò la dura operazione di tuning del sistema operativo dovendo installare Apache con modulo PHP-MySql, e MySql.

Ebbi frustranti problemi a configurare il modem e a installare i programmi che mi servivano, la maggior difficoltà fu individuare le informazioni che mi spiegassero come eseguire ciò che mi interessava. Fui letteralmente sommerso di HOW-TO e guide tecniche che avrebbero dovuto risolvere i miei problemi, purtroppo ognuna risolveva il problema in maniera diversa da tutte le altre e puntualmente facevano riferimento a file di setting delle impostazioni che non trovavo sul mio sistema. Venni a capo dei vari problemi a volte semplicemente guardando con più attenzione la documentazione a volte entrando nel merito dei programmi di installazione e modificandoli secondo necessità.

Ebbi dalla mia parte i mezzi per apprendere il funzionamento del sistema almeno per le parti nelle quali ho dovuto intervenire, ma un utente meno “in-

formatizzato" avrebbe avuto le stesse possibilità? Ritengo di no. Però bisogna anche chiedersi: un utente meno informatizzato avrebbe avuto la necessità di installare quei programmi? Ritengo di no. Quel che garantisce Linux è la possibilità di risolvere ogni problema di configurazione, previo il giusto tributo di notti insonni per capire dove mettere le mani. Il problema sembra paradossale ma può essere sintetizzato così : le possibilità di scelta sono talmente tante da diventare troppe e finiscono per confondere chi si avvicina allo strumento. Le interfacce grafiche sono realmente molto potenti a livello di personalizzazione e di intervento sul sistema ma non sono organiche, col risultato di confondere chi ha bisogno di personalizzazioni non banali. Un altro fattore di complicazione riguarda le diversità tra le versioni in circolazione e la relativa manualistica: Linux è uno ma i file di configurazione cambiano nome e posizione da una versione all'altra rendendo obsoleti e/o inutili suggerimenti e manuali. Chiaramente le modifiche sono sempre riportate nei "readme.txt" che accompagnano il software, ma si tratta di migliaia di righe non facilmente consultabili da chi non ha le idee molto chiare sulle operazioni che deve eseguire. D'altro canto esiste una poderosa documentazione su praticamente tutto ciò che può accadere su una macchina gestita da Linux e quindi questo potrebbe limitare i vari problemi a un esercizio di pazienza e di ricerca.

Comunque alla fine riuscii ad avere una macchina completamente operativa e specchio virtuale del server che avrebbe supportato il progetto.

4.3 Il progetto

L'analisi delle specifiche e degli esempi di scheda di inserimento dati forniti portarono a un diagramma E-R con le seguenti tabelle:

- anagrafica_utenti (user_id, dati anagrafici del ricercatore...).
- anagrafica_batteri (microrganismo, test specifici...)
- anagrafica_antibiotici (antibiotico, microrganismo, valori ammissibili e non...)
- anagrafica_scheda (user_id, id_scheda, microrganismo, id_assoluto, test_specifico, dati_paziente...)
- scheda (user_id, id_scheda, antibiotico, id_assoluto, valore...)

In realtà le tabelle sono di più per via del fatto che il progetto SEPRA comprende più test diversi, la struttura riportata sopra è comunque generalmente valida.

La tabella anagrafica_utenti contiene tutti i dati relativi al personale abilitato ad accedere al servizio. Il campo user_id è uguale alla user con cui gli utenti si connettono al servizio.

La tabella anagrafica_batteri riporta i dati necessari a trattare specificatamente ciascun microorganismo. La tabella anagrafica_antibiotici contiene le informazioni riguardanti il test per ogni antibiotico. Le tabelle relative alla scheda riportano i dati relativi a ciascuna scheda, sono vincolate dalla chiave user_id e id_scheda. È stato necessario inserire il campo id_assoluto per una duplice gestione della visualizzazione delle schede. Ricordo infatti che i ricercatori possono consultare solo le proprie schede e pertanto le identificano mediante l’ordine di inserimento (il campo id_scheda), altri utenti possono visionare tutte le schede ma non possono sapere chi le ha inserite e queste vengono loro presentate con un ordinale assoluto (id_assoluto), utile anche ai propri fini di catalogazione e elaborazione.

Le pagine sono state costruite in HTML e presentate all’utente, interpretati i suggerimenti e conseguita l’approvazione si è proseguito a sostituire tutto il possibile con il linguaggio di scripting PHP3. Il PHP3, venendo interpretato da un modulo apposito del Web Server Apache, permette la generazione di HTML condizionata. Ossia si può determinare, esaminando la richiesta e l’identità dell’utente, la struttura e le funzionalità della pagina richiesta. Ad esempio il bottone che permette di scaricare il file contenente i dati via e-mail (in allegato) è visualizzato solo per gli utenti abilitati alla funzionalità, la presentazione delle schede viene automaticamente limitata a quelle inserite dall’utente o estesa a tutte le schede a seconda di chi richiede la visualizzazione. Questo ha permesso di ingegnerizzare le pagine, ossia si mantiene una struttura comune scritta in HTML e il resto del contenuto viene determinato dal PHP3 a seconda della situazione.

Il PHP3 ha fornito anche un’ottima prova di integrazione con il database MySql permettendo la gestione di recordset e di connessioni permanenti all’interno della stessa pagina.

4.3.1 La sicurezza

Il discorso sicurezza è stato affrontato con particolare attenzione essendo una richiesta specifica dell'utente. L'utente non era interessato a proteggere la connessione da intercettazioni sulle linee di trasmissione quanto a fare in modo che i ricercatori non autorizzati, o chi per essi, potesse visionare l'intera mole di dati raccolti.

Si è così optato per la protezione dei dati basata sul database, in questo modo:

1. L'utente si connette al sito e alla pagina di benvenuto inserisce user e password

2. Il sistema si connette con il database e verifica che i dati di riconoscimento inseriti siano validi

3. Se l'utente è autorizzato ad accedere al sistema gli viene associato un token che utilizzerà, inconsapevolmente in quanto è gestito interamente dal sistema, per usufruire dei servizi.

4. Il token è associato alla connessione, viene disattivato dopo un certo periodo in cui non sono effettuate operazioni (la connessione "muore") o quando l'utente decide di eseguire il logout.

5. Nel caso, scaduto il token, qualcuno cerchi di utilizzare le pagine caricate nella memoria del browser per navigare il sito verrà educatamente, ma fermamente, invitato a desistere.

Il token, così come le coppie user-password, sono memorizzate su apposite tabelle del database.

Lo stesso tipo di sicurezza si sarebbe potuto ottenere con l'utilizzo dei cookie via Internet. Non è stata scelta questa modalità per alcuni problemi relativi alla loro gestione: non tutti i browser li sanno gestire e non tutti i browser li gestiscono allo stesso modo. Riferendoci a un'utenza con scarsa dimestichezza nel campo informatico e difficilmente raggiungibile in caso di richieste di supporto si è preferito accentrare sul server ogni aspetto "gestionale".

Il fattore sicurezza è incentrato sul fatto che l'accesso è controllato dal database che permette interrogazioni solo a livello locale (o localhost) e non

in modalità remota. Quindi l'accesso al sito può avvenire solo tramite un processo innescato sul server stesso. Il database è a sua volta protetto, oltre che da meccanismi propri, dalle procedure di login sicuro del sistema operativo. Non si è ritenuto necessario ricorrere ad ulteriori meccanismi di protezione.

4.4 L'esito

Il progetto (locato all'indirizzo: smb.webline.it) è stato gradito dalla clientela e la raccolta dati è incominciata da poco (Maggio 2000), al momento non sono ancora pervenuti messaggi all'indirizzo di posta a disposizione degli utenti per inviare richieste di aiuto, di conseguenza sembra che l'interfaccia sia perlomeno intuitivamente funzionale.

Riguardo gli strumenti utilizzati non posso che essere soddisfatto: Apache merita tutto il successo che sta avendo, MySql è un'ottima scelta nel caso di database relazionali per moli di dati medie e piccole (spero vivamente che PostGreSql sia reso pienamente funzionale perché ha sicuramente ottime prospettive), inoltre negli ultimi tempi sono nate anche delle interfacce grafiche sullo stile del noto Access Microsoft. PHP3 è stata la vera manna: semplice, ricco di funzionalità, potente per espressività e soprattutto molto veloce nell'esecuzione. L'unico problema riguarda il fatto che deve essere ancora esteso per gestire progetti di grandi dimensioni, infatti il supporto agli oggetti è buono ma non esaustivo. Uscirà a breve la versione PHP4 interamente basata su tecnologia ad oggetti ed è in cantiere la versione PHP5 che a detta dei responsabili sarà dedicata alla gestione di grandi e complessi siti: dichiarano che vorranno concorrere con JAVA!

Problemi di installazione a parte è tutto filato liscio. La scelta Open Source si è rivelata vincente e gli stessi strumenti saranno impiegati nello sviluppo di altri siti dedicati alla raccolta di dati per la ricerca medica.

5 Conclusioni

Nel corpo della tesi si è avuto modo di tracciare la storia dei notevoli risultati raggiunti in tempi relativamente brevi dal software Open Source. Tali risultati sono frutto di alcuni importanti fattori di carattere generale non strettamente legati al mondo dell'informatica ma, piuttosto, al modo di trattare l'informazione. Fattori questi che si è già avuto modo di notare in altre epoche storiche, epoche contraddistinte da un grande sviluppo culturale.

In generale si può affermare che la condivisione dell'informazione crea informazione in un circolo virtuoso che ha come risultato il progresso (e non la semplice evoluzione) della conoscenza. La differenza tra progresso e evoluzione è sottile eppure fondamentale. L'evoluzione costituisce un migliore adattamento delle capacità di difesa o di attacco nei confronti di una minaccia (e quindi un'intrinseca maggiore probabilità di sopravvivenza), il progresso permette di superare il problema annullando la minaccia. In termini informatici possiamo sostenere che lo sviluppo dell'interfaccia grafica è stato il progresso che ha permesso l'abbandono delle interfacce testuali divenute eccessivamente complicate, la continua elaborazione dell'interfaccia (più semplice, più potente) è invece un processo evolutivo che tende ad adattare la facile gestione richiesta dall'utente alla complessità crescente del sistema da controllare.

Storicamente possiamo riconoscere alcuni periodi contrassegnati da una forte spinta al progresso, traccerò le linee fondamentali di due di questi periodi al fine di trovare degli elementi in comune con le basi metodologiche dell'Open Source.

Dal V secolo A.C. in una regione europea conosciuta come Grecia si sviluppò la cultura del pensiero speculativo, cioè il pensiero non atto a risolvere problemi di mera sussistenza ma a fornire risposte alle domande che già allora l'uomo si poneva sulla propria esistenza. Nacque la filosofia, l'amore per il sapere, nelle branche della fisica, della matematica e della ricerca delle origini di questo desiderio di conoscere. Nacquero le scuole di filosofia guidate dai

maestri del pensiero, uomini la cui capacità d'indagine era riconosciuta come superiore e ammirevole dai propri contemporanei, personalità ancora oggi oggetto di studio e di riferimento come Socrate, Platone, Aristotele. L'opera di questi uomini fu concepire metodi di indagine innovativi e non legati al campo di applicazione. Studio scientifico e umanistico erano fusi in un unicum che aveva come scopo la ricerca del vero, uno stato superiore della conoscenza oltre l'esperibile. Formularono criteri d'indagine e strumenti logici. Fondarono le basi del pensiero occidentale.

Tale periodo di grande sviluppo della conoscenza fu reso possibile da alcune condizioni sociali. In primo luogo possiamo riconoscere un grande sviluppo economico e un conseguente innalzamento del livello di ricchezza. Questo permise, perlomeno ad alcuni, di distaccarsi dall'immediata preoccupazione della propria sopravvivenza e di occuparsi di problemi meno pressanti come l'origine e il termine dell'esistenza, stabilire cosa è bello, dedicarsi ai piaceri dell'arte. Ancora più importante fu la possibilità di poter sovvenzionare persone che si occupassero interamente di rispondere alla nascente necessità di conoscere. Nacquero le figure del pensatore di professione e dell'artista, che benché non scambiassero beni di prima necessità erano comunque retribuite per la fornitura di un valore aggiunto difficilmente quantificabile.

Un'altra condizione che permise uno sviluppo così rapido fu la relativa facilità di spostamento di persone e beni, e quindi di informazioni. La fitta rete mercantile collegava le poleis trasportando non solo beni ma anche notizie con un ritmo fino ad allora mai raggiunto. Le idee viaggiavano alla massima velocità consentita (quella delle navi) propagandosi senza un controllo organizzato. Si era ben lontani dal concetto di proprietà intellettuale e questo permise una continua elaborazione di quanto si veniva a conoscenza.

Quindi possiamo notare alcuni fattori sostanziali: surplus economico e libera diffusione delle idee alla massima velocità consentita.

Un altro periodo di splendore si raggiunse nel Rinascimento (XV-XVI sec.) in Italia e che influenzò per tutto il secolo successivo la regione europea. Anche in quel periodo riscontriamo un surplus economico che permette di investire in un surplus di beni. È il periodo dei grandi artisti e dei grandi scienziati, che sovvenzionati e contesi da banchieri, mercanti e soprattutto governi poterono dedicarsi interamente alle proprie ambizioni. E in quel periodo nacque un nuovo e potente mezzo per la diffusione delle idee: la stampa a caratteri mobili ideata da Gutemberg (1438). Fino ad allora il propagarsi

delle informazioni era stato in qualche modo rallentato dal lento e complesso (e quindi molto costoso) metodo di trascrizione dei documenti. Gli amanuensi impiegavano anche più di un anno a copiare un'opera, di conseguenza la priorità nella scelta dei testi da copiare era riservata ai testi classici e religiosi. La diffusione delle nuove idee era frenata dal fatto che nessuno avrebbe investito tempo e risorse in opere delle quali non era possibile avere una universale prova di validità (o meglio: di accettazione).

La situazione cambiò con l'invenzione di Gutemberg. Riprodurre l'informazione fu molto più economico e rapido e così poterono essere divulgate opere giovani e innovatrici. Le idee tornarono a circolare e a produrre altre idee, gli scienziati poterono trovare conferme o smentite in altri studiosi poco raggiungibili direttamente. La base di conoscenza disponibile divenne più ampia e più solida consentendo il raggiungimento di nuovi obbiettivi.

Ancora una volta la possibilità di creare informazione associata con la possibilità di diffonderla velocemente furono il propellente del progresso. È infatti del 1687 la pubblicazione dei "Principi matematici della filosofia naturale" di sir Isaac Newton. Opera fondamentale per il progresso delle scienze fisiche e matematiche.

L'Open Source si è avvalso degli stessi fattori presenti negli esempi riportati. Nei primi anni dello sviluppo informatico università e governi sovvenzionarono le attività di ricerca, subito seguiti da investitori privati, il surplus economico fu investito in attività di ricerca. Il primo risultato fu la creazione di tecnologie che crearono nuove possibilità di sviluppo. Lo sviluppo fu incentivato dal fatto che la maggior parte delle innovazioni erano a disposizione di tutti i ricercatori perché potessero continuare a innovare e a verificare. Si ha ancora qui un elemento riscontrato negli esempi precedenti: la condivisione delle informazioni. Il mezzo di comunicazione su cui viaggiavano dati, prove e idee fu fornito dalla stessa nascente tecnologia: le reti di comunicazione tra computer. L'evoluzione di quelle reti ha portato a Internet, oggi la rete più usata per ogni tipo di comunicazione. Ed è proprio mediante Internet che gli hacker hanno potuto organizzarsi e collaborare a grandi progetti, portandoli a termine. Internet ha consentito una capillare distribuzione della conoscenza in tempi adeguatamente rapidi. La libertà di accesso al mezzo di comunicazione e i relativi bassi costi dello sviluppo informatico hanno permesso un ulteriore grado di libertà dai sovvenzionatori che diventano sempre più clienti. Lo sviluppo è diventato quindi un'attività pressoché indipendente e libera.

L'Open Source ha ottenuto grandi risultati in tempi rapidi in quanto adotta il modello di sviluppo più rapido e sicuro che si conosca: la libera informazione. Tale modello è incredibilmente semplice ed economico ma viene adottato raramente per avvallare altri interessi economici basati sul diritto di possesso e quindi di vendita. Sostenere l'Open Source non significa combattere il mercato (anzi: l'OS è nato con lo scopo di abbracciare il mercato, a differenza del Free Software), significa puntare su un veloce mezzo di sviluppo: l'unico in grado di colmare il gap nell'innovazione tecnologica che ci separa dai leader USA e Giappone.

Come gestire la locomotiva Open Source? Facciamo tesoro della lezione Linux/Emacs! Torvalds ha adottato un metodo, Raymond ha dimostrato che non ha funzionato per pura fortuna. Entrambi sono partiti con un collegamento Internet e il PC di casa. Che risultati potrebbe ottenere un'istituzione che patrocini un progetto di catalogazione, valutazione e sviluppo del patrimonio Open Source oggi presente?

Il Prof. Angelo Meo (Politecnico di Torino) ha delineato nel Febbraio 1998 [1] un lodevole e ambizioso progetto, il progetto "Freeware", con il fine che egli stesso esplicita:

> "L'obiettivo centrale del progetto qui proposto e' [...] rappresentato dalla sistematizzazione organica, a fini industriali, comprensiva del lavoro di certificazione e di ampliamento, ove necessario, del materiale disponibile. Il sogno è fare dell'Italia la capitale mondiale del "freeware". Quattro aree di attività, concettualmente sequenziali ma praticamente interallacciate, caratterizzeranno il lavoro da svolgere"

Le aree coprono aspetti di documentazione, valutazione e sviluppo. Nel progetto si prevede la collaborazione di produttori industriali soprattutto per quanto riguarda l'area dello sviluppo. Nel documento è riportata anche una previsione dei costi e dei tempi necessari per le diverse aree di lavoro.

Ritengo il progetto, come delineato, molto valido e accurato soprattutto nella valutazione del ritorno economico su scala nazionale (oggi forse da ritoccare ancora più al rialzo) e la sua valenza per quanto riguarda l'economia solidale verso i paesi del terzo mondo. Sarebbe certamente un progetto a 360 gradi, sferici.

Quello che auspica il Prof. Meo per varare il progetto è la creazione di una struttura organizzativa in termini istituzionali. Ma poiché parliamo di Open

Source perché non adottarlo fin dall'inizio? Torvalds, Raynolds, Stallman e tanti altri sono riusciti e perseguono a guidare progetti di successo senza necessità di apparati direzionali. Raymond ha spiegato (in questa tesi riportato al capitolo 3) come impostare un progetto Open Source: fornire un nucleo comprensibile come punto di partenza, presentare un intento chiaro, sfruttare i suggerimenti, utilizzare un pizzico di carisma. Tutto il resto è creazione. Se questa volta a partire fosse un'Università? Che impulso potrebbero fornire studenti didatticamente impiegati nella documentazione e nello sviluppo? La diretta "sorveglianza" di professori di ingegneria del software? Il poter disporre di capacità di archiviazione e di raccolta documenti? Il poter disporre di una struttura stabile e permanente come punto di riferimento?

Le possibilità sarebbero vastissime e partire presto significherebbe diventare il polo europeo del software, il centro catalizzatore di sviluppo e di raccolta. Inoltre il progetto potrebbe estendersi a una componente essenziale per lo sviluppo di nuove tecnologie: l'hardware, e questa potrebbe essere la vera grande novità, per una volta in questo settore, tutta europea.



Le licenze Open Source

In questa appendice sono raccolte alcune licenze omologate “OSI Certified“ dalla Open Source Initiative. Riporto le licenze in lingua originale, quindi nella formulazione valida a fini legali. La Open Source Initiative è disponibile a omologare nuove licenze Open Source ma consiglia caldamente di avvalersi di quelli già esistenti.

A.1 General Purpose License

Version 2, June 1991

Copyright (C) 1989, 1991 Free Software Foundation, Inc. 675 Mass Ave, Cambridge, MA 02139, USA. Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

A.1.1 Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software–to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation’s software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Library General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

A.1.2 GNU GENERAL PUBLIC LICENSE

TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another langua-

ge. (Hereinafter, translation is included without limitation in the term “modification”.) Each licensee is addressed as “you”.

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute verbatim copies of the Program’s source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

   You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

   a) You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.

   b) You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.

   c) If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the

user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

    a) Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

    b) Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

c) Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

   If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

   It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

   This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copy-

right holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

9. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

   Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

A.1.3 NO WARRANTY

1. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND

PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

2. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

A.1.4 END OF TERMS AND CONDITIONS

A.2 GNU Lesser General Purpose License

Version 2.1, February 1999

Copyright (C) 1991, 1999 Free Software Foundation, Inc.59 Temple Place, Suite 330, Boston, MA 02111-1307 USAEveryone is permitted to copy and distribute verbatim copiesof this license document, but changing it is not allowed. [This is the first released version of the Lesser GPL. It also counts as the successor of the GNU Library Public License, version 2, hence the version number 2.1.]

A.2.1 Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public Licenses are intended to guarantee your freedom to share and change free software–to make sure the software is free for all its users.

This license, the Lesser General Public License, applies to some specially designated software packages–typically libraries–of the Free Software Foundation and other authors who decide to use it. You can use it too, but we suggest you first think carefully about whether this license or the ordinary General Public License is the better strategy to use in any particular case, based on the explanations below.

When we speak of free software, we are referring to freedom of use, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish); that you receive source code or can get it if you want it; that you can change the software and use pieces of it in new free programs; and that you are informed that you can do these things.

To protect your rights, we need to make restrictions that forbid distributors to deny you these rights or to ask you to surrender these rights. These restrictions translate to certain responsibilities for you if you distribute copies of the library or if you modify it.

For example, if you distribute copies of the library, whether gratis or for a fee, you must give the recipients all the rights that we gave you. You must make sure that they, too, receive or can get the source code. If you link other code with the library, you must provide complete object files to the recipients, so that they can relink them with the library after making changes to the library and recompiling it. And you must show them these terms so they know their rights.

We protect your rights with a two-step method: (1) we copyright the library, and (2) we offer you this license, which gives you legal permission to copy, distribute and/or modify the library.

To protect each distributor, we want to make it very clear that there is no warranty for the free library. Also, if the library is modified by someone else and passed on, the recipients should know that what they have is not the original version, so that the original author's reputation will not be affected by problems that might be introduced by others.

Finally, software patents pose a constant threat to the existence of any free program. We wish to make sure that a company cannot effectively restrict the users of a free program by obtaining a restrictive license from a patent holder. Therefore, we insist that any patent license obtained for a version of the library must be consistent with the full freedom of use specified in this

license.

Most GNU software, including some libraries, is covered by the ordinary GNU General Public License. This license, the GNU Lesser General Public License, applies to certain designated libraries, and is quite different from the ordinary General Public License. We use this license for certain libraries in order to permit linking those libraries into non-free programs.

When a program is linked with a library, whether statically or using a shared library, the combination of the two is legally speaking a combined work, a derivative of the original library. The ordinary General Public License therefore permits such linking only if the entire combination fits its criteria of freedom. The Lesser General Public License permits more lax criteria for linking other code with the library.

We call this license the “Lesser” General Public License because it does Less to protect the user’s freedom than the ordinary General Public License. It also provides other free software developers Less of an advantage over competing non-free programs. These disadvantages are the reason we use the ordinary General Public License for many libraries. However, the Lesser license provides advantages in certain special circumstances.

For example, on rare occasions, there may be a special need to encourage the widest possible use of a certain library, so that it becomes a de-facto standard. To achieve this, non-free programs must be allowed to use the library. A more frequent case is that a free library does the same job as widely used non-free libraries. In this case, there is little to gain by limiting the free library to free software only, so we use the Lesser General Public License.

In other cases, permission to use a particular library in non-free programs enables a greater number of people to use a large body of free software. For example, permission to use the GNU C Library in non-free programs enables many more people to use the whole GNU operating system, as well as its variant, the GNU/Linux operating system.

Although the Lesser General Public License is Less protective of the users’ freedom, it does ensure that the user of a program that is linked with the Library has the freedom and the wherewithal to run that program using a modified version of the Library.

The precise terms and conditions for copying, distribution and modification follow. Pay close attention to the difference between a “work based on the library” and a “work that uses the library”. The former contains code de-

rived from the library, whereas the latter must be combined with the library in order to run.

A.2.2 LGPL TERMS AND CONDITIONS

0. This License Agreement applies to any software library or other program which contains a notice placed by the copyright holder or other authorized party saying it may be distributed under the terms of this Lesser General Public License (also called “this License”). Each licensee is addressed as “you”.

   A “library” means a collection of software functions and/or data prepared so as to be conveniently linked with application programs (which use some of those functions and data) to form executables.

   The “Library”, below, refers to any such software library or work which has been distributed under these terms. A “work based on the Library” means either the Library or any derivative work under copyright law: that is to say, a work containing the Library or a portion of it, either verbatim or with modifications and/or translated straightforwardly into another language. (Hereinafter, translation is included without limitation in the term “modification”.)

   “Source code” for a work means the preferred form of the work for making modifications to it. For a library, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the library.

   Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running a program using the Library is not restricted, and output from such a program is covered only if its contents constitute a work based on the Library (independent of the use of the Library in a tool for writing it). Whether that is true depends on what the Library does and what the program that uses the Library does.

   You may copy and distribute verbatim copies of the Library’s complete source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copy-

right notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and distribute a copy of this License along with the Library.

1. You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Library or any portion of it, thus forming a work based on the Library, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

   a) The modified work must itself be a software library

   b) You must cause the files modified to carry prominent notices stating that you changed the files and the date of any change.

   c) You must cause the whole of the work to be licensed at no charge to all third parties under the terms of this License.

   d) If a facility in the modified Library refers to a function or a table of data to be supplied by an application program that uses the facility, other than as an argument passed when the facility is invoked, then you must make a good faith effort to ensure that, in the event an application does not supply such function or table, the facility still operates, and performs whatever part of its purpose remains meaningful.

      (For example, a function in a library to compute square roots has a purpose that is entirely well-defined independent of the application. Therefore, Subsection 2d requires that any application-supplied function or table used by this function must be optional: if the application does not supply it, the square root function must still compute square roots.)

      These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Library, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the

Library, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Library.

In addition, mere aggregation of another work not based on the Library with the Library (or with a work based on the Library) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may opt to apply the terms of the ordinary GNU General Public License instead of this License to a given copy of the Library. To do this, you must alter all the notices that refer to this License, so that they refer to the ordinary GNU General Public License, version 2, instead of to this License. (If a newer version than version 2 of the ordinary GNU General Public License has appeared, then you can specify that version instead if you wish.) Do not make any other change in these notices.

   Once this change is made in a given copy, it is irreversible for that copy, so the ordinary GNU General Public License applies to all subsequent copies and derivative works made from that copy.

   This option is useful when you wish to copy part of the code of the Library into a program that is not a library.

4. You may copy and distribute the Library (or a portion or derivative of it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange.

   If distribution of object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place satisfies the requirement to distribute the source code, even though third parties are not compelled to copy the source along with the object code.

5. A program that contains no derivative of any portion of the Library, but is designed to work with the Library by being compiled or linked with it, is called a "work that uses the Library". Such a work, in isolation, is not a derivative work of the Library, and therefore falls outside the scope of this License.

   However, linking a "work that uses the Library" with the Library creates an executable that is a derivative of the Library (because it contains portions of the Library), rather than a "work that uses the library". The executable is therefore covered by this License. Section 6 states terms for distribution of such executables.

   When a "work that uses the Library" uses material from a header file that is part of the Library, the object code for the work may be a derivative work of the Library even though the source code is not. Whether this is true is especially significant if the work can be linked without the Library, or if the work is itself a library. The threshold for this to be true is not precisely defined by law.

   If such an object file uses only numerical parameters, data structure layouts and accessors, and small macros and small inline functions (ten lines or less in length), then the use of the object file is unrestricted, regardless of whether it is legally a derivative work. (Executables containing this object code plus portions of the Library will still fall under Section 6.)

   Otherwise, if the work is a derivative of the Library, you may distribute the object code for the work under the terms of Section 6. Any executables containing that work also fall under Section 6, whether or not they are linked directly with the Library itself.

6. As an exception to the Sections above, you may also combine or link a "work that uses the Library" with the Library to produce a work containing portions of the Library, and distribute that work under terms of your choice, provided that the terms permit modification of the work for the customer's own use and reverse engineering for debugging such modifications.

   You must give prominent notice with each copy of the work that the Library is used in it and that the Library and its use are covered by this

License. You must supply a copy of this License. If the work during execution displays copyright notices, you must include the copyright notice for the Library among them, as well as a reference directing the user to the copy of this License. Also, you must do one of these things:

a) Accompany the work with the complete corresponding machine-readable source code for the Library including whatever changes were used in the work (which must be distributed under Sections 1 and 2 above); and, if the work is an executable linked with the Library, with the complete machine-readable "work that uses the Library", as object code and/or source code, so that the user can modify the Library and then relink to produce a modified executable containing the modified Library. (It is understood that the user who changes the contents of definitions files in the Library will not necessarily be able to recompile the application to use the modified definitions.)

b) Use a suitable shared library mechanism for linking with the Library. A suitable mechanism is one that (1) uses at run time a copy of the library already present on the user's computer system, rather than copying library functions into the executable, and (2) will operate properly with a modified version of the library, if the user installs one, as long as the modified version is interface-compatible with the version that the work was made with.

c) Accompany the work with a written offer, valid for at least three years, to give the same user the materials specified in Subsection 6a, above, for a charge no more than the cost of performing this distribution.

d) If distribution of the work is made by offering access to copy from a designated place, offer equivalent access to copy the above specified materials from the same place.

e) Verify that the user has already received a copy of these materials or that you have already sent this user a copy.

For an executable, the required form of the "work that uses the Library" must include any data and utility programs needed for reproducing the executable from it. However, as a special exception, the

materials to be distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

It may happen that this requirement contradicts the license restrictions of other proprietary libraries that do not normally accompany the operating system. Such a contradiction means you cannot use both them and the Library together in an executable that you distribute.

7. You may place library facilities that are a work based on the Library side-by-side in a single library together with other library facilities not covered by this License, and distribute such a combined library, provided that the separate distribution of the work based on the Library and of the other library facilities is otherwise permitted, and provided that you do these two things:

   a) Accompany the combined library with a copy of the same work based on the Library, uncombined with any other library facilities. This must be distributed under the terms of the Sections above.

   b) Give prominent notice with the combined library of the fact that part of it is a work based on the Library, and explaining where to find the accompanying uncombined form of the same work.

8. You may not copy, modify, sublicense, link with, or distribute the Library except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, link with, or distribute the Library is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

9. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Library or its derivative works. These actions are prohibited by law

if you do not accept this License. Therefore, by modifying or distributing the Library (or any work based on the Library), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Library or works based on it.

10. Each time you redistribute the Library (or any work based on the Library), the recipient automatically receives a license from the original licensor to copy, distribute, link with or modify the Library subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties with this License.

11. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Library at all. For example, if a patent license would not permit royalty-free redistribution of the Library by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Library.

    If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply, and the section as a whole is intended to apply in other circumstances.

    It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

12. If the distribution and/or use of the Library is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Library under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

13. The Free Software Foundation may publish revised and/or new versions of the Lesser General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

    Each version is given a distinguishing version number. If the Library specifies a version number of this License which applies to it and “any later version”, you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Library does not specify a license version number, you may choose any version ever published by the Free Software Foundation.

14. If you wish to incorporate parts of the Library into other free programs whose distribution conditions are incompatible with these, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

A.2.3 NO WARRANTY

15. BECAUSE THE LIBRARY IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE LIBRARY, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR

OTHER PARTIES PROVIDE THE LIBRARY "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE LIBRARY IS WITH YOU. SHOULD THE LIBRARY PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

16. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE LIBRARY AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE LIBRARY (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE LIBRARY TO OPERATE WITH ANY OTHER SOFTWARE), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

A.2.4 END OF TERMS AND CONDITIONS

A.3 BSD License

Copyright (c) <YEAR>, <OWNER> All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

- Neither name of the <ORGANIZATION> nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS “AS IS” AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE REGENTS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

A.4 MIT License (X Consortium)

Copyright (c) <year> <copyright holders>

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the “Software”), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED “AS IS”, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN

ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

A.5 The Artistic License

A.5.1 Preamble

The intent of this document is to state the conditions under which a Package may be copied, such that the Copyright Holder maintains some semblance of artistic control over the development of the package, while giving the users of the package the right to use and distribute the Package in a more-or-less customary fashion, plus the right to make reasonable modifications.

A.5.2 Definitions:

- “Package” refers to the collection of files distributed by the Copyright Holder, and derivatives of that collection of files created through textual modification.
- “Standard Version” refers to such a Package if it has not been modified, or has been modified in accordance with the wishes of the Copyright Holder.
- “Copyright Holder” is whoever is named in the copyright or copyrights for the package.
- “You” is you, if you’re thinking about copying or distributing this Package.
- “Reasonable copying fee” is whatever you can justify on the basis of media cost, duplication charges, time of people involved, and so on. (You will not be required to justify it to the Copyright Holder, but only to the computing community at large as a market that must bear the fee.)
- “Freely Available” means that no fee is charged for the item itself, though there may be fees involved in handling the item. It also means that

recipients of the item may redistribute it under the same conditions they received it.

1. You may make and give away verbatim copies of the source form of the Standard Version of this Package without restriction, provided that you duplicate all of the original copyright notices and associated disclaimers.

2. You may apply bug fixes, portability fixes and other modifications derived from the Public Domain or from the Copyright Holder. A Package modified in such a way shall still be considered the Standard Version.

3. You may otherwise modify your copy of this Package in any way, provided that you insert a prominent notice in each changed file stating how and when you changed that file, and provided that you do at least ONE of the following:

   a) place your modifications in the Public Domain or otherwise make them Freely Available, such as by posting said modifications to Usenet or an equivalent medium, or placing the modifications on a major archive site such as ftp.uu.net, or by allowing the Copyright Holder to include your modifications in the Standard Version of the Package.

   b) use the modified Package only within your corporation or organization.

   c) rename any non-standard executables so the names do not conflict with standard executables, which must also be provided, and provide a separate manual page for each non-standard executable that clearly documents how it differs from the Standard Version.

   d) make other distribution arrangements with the Copyright Holder.

4. You may distribute the programs of this Package in object code or executable form, provided that you do at least ONE of the following:

   a) distribute a Standard Version of the executables and library files, together with instructions (in the manual page or equivalent) on where to get the Standard Version.

b) accompany the distribution with the machine-readable source of the Package with your modifications.

c) accompany any non-standard executables with their corresponding Standard Version executables, giving the non-standard executables non-standard names, and clearly documenting the differences in manual pages (or equivalent), together with instructions on where to get the Standard Version.

d) make other distribution arrangements with the Copyright Holder.

5. You may charge a reasonable copying fee for any distribution of this Package. You may charge any fee you choose for support of this Package. You may not charge a fee for this Package itself. However, you may distribute this Package in aggregate with other (possibly commercial) programs as part of a larger (possibly commercial) software distribution provided that you do not advertise this Package as a product of your own.

6. The scripts and library files supplied as input to or produced as output from the programs of this Package do not automatically fall under the copyright of this Package, but belong to whomever generated them, and may be sold commercially, and may be aggregated with this Package.

7. C or perl subroutines supplied by you and linked into this Package shall not be considered part of this Package.

8. The name of the Copyright Holder may not be used to endorse or promote products derived from this software without specific prior written permission.

9. THIS PACKAGE IS PROVIDED "AS IS" AND WITHOUT ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, WITHOUT LIMITATION, THE IMPLIED WARRANTIES OF MERCHANTIBILITY AND FITNESS FOR A PARTICULAR PURPOSE.

A.5.3 The End

A.6 The zlib/libpng License

Copyright (c) <year> <copyright holders>

This software is provided ’as-is’, without any express or implied warranty. In no event will the authors be held liable for any damages arising from the use of this software.

Permission is granted to anyone to use this software for any purpose, including commercial applications, and to alter it and redistribute it freely, subject to the following restrictions:

1. The origin of this software must not be misrepresented; you must not claim that you wrote the original software. If you use this software in a product, an acknowledgment in the product documentation would be appreciated but is not required.

2. Altered source versions must be plainly marked as such, and must not be misrepresented as being the original software.

3. This notice may not be removed or altered from any source distribution.

A.7 Mozilla Public License

A.7.1 Definitions

1. “Contributor” means each entity that creates or contributes to the creation of Modifications.

2. “Contributor Version” means the combination of the Original Code, prior Modifications used by a Contributor, and the Modifications made by that particular Contributor.

3. “Covered Code” means the Original Code or Modifications or the combination of the Original Code and Modifications, in each case including portions thereof.

4. “Electronic Distribution Mechanism” means a mechanism generally accepted in the software development community for the electronic transfer of data.

5. “Executable” means Covered Code in any form other than Source Code.

6. “Initial Developer” means the individual or entity identified as the Initial Developer in the Source Code notice required by Exhibit A.

7. “Larger Work” means a work which combines Covered Code or portions thereof with code not governed by the terms of this License.

8. “License” means this document.

9. “Modifications” means any addition to or deletion from the substance or structure of either the Original Code or any previous Modifications. When Covered Code is released as a series of files, a Modification is:

   A. Any addition to or deletion from the contents of a file containing Original Code or previous Modifications.

   B. Any new file that contains any part of the Original Code or previous Modifications.

10. “Original Code” means Source Code of computer software code which is described in the Source Code notice required by Exhibit A as Original Code, and which, at the time of its release under this License is not already Covered Code governed by this License.

11. “Source Code” means the preferred form of the Covered Code for making modifications to it, including all modules it contains, plus any associated interface definition files, scripts used to control compilation and installation of an Executable, or a list of source code differential comparisons against either the Original Code or another well known, available Covered Code of the Contributor’s choice. The Source Code can be in a compressed or archival form, provided the appropriate decompression or de-archiving software is widely available for no charge.

12. "You" means an individual or a legal entity exercising rights under, and complying with all of the terms of, this License or a future version of this License issued under Section 6.1. For legal entities, "You" includes any entity which controls, is controlled by, or is under common control with You. For purposes of this definition, "control" means (a) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (b) ownership of fifty percent (50

A.7.2 Source Code License

The Initial Developer Grant

The Initial Developer hereby grants You a world-wide, royalty-free, non-exclusive license, subject to third party intellectual property claims:

a) to use, reproduce, modify, display, perform, sublicense and distribute the Original Code (or portions thereof) with or without Modifications, or as part of a Larger Work; and

b) under patents now or hereafter owned or controlled by Initial Developer, to make, have made, use and sell ("Utilize") the Original Code (or portions thereof), but solely to the extent that any such patent is reasonably necessary to enable You to Utilize the Original Code (or portions thereof) and not to any greater extent that may be necessary to Utilize further Modifications or combinations.

Contributor Grant

Each Contributor hereby grants You a world-wide, royalty-free, non-exclusive license, subject to third party intellectual property claims:

a) to use, reproduce, modify, display, perform, sublicense and distribute the Modifications created by such Contributor (or portions thereof) either on an unmodified basis, with other Modifications, as Covered Code or as part of a Larger Work; and

b) under patents now or hereafter owned or controlled by Contributor, to Utilize the Contributor Version (or portions thereof), but solely to the extent that any such patent is reasonably necessary to enable You to Utilize the Contributor Version (or portions thereof), and not to any greater extent that may be necessary to Utilize further Modifications or combinations.

A.7.3 Distribution Obligations

Application of License.

The Modifications which You create or to which You contribute are governed by the terms of this License, including without limitation Section 2.2. The Source Code version of Covered Code may be distributed only under the terms of this License or a future version of this License released under Section 6.1, and You must include a copy of this License with every copy of the Source Code You distribute. You may not offer or impose any terms on any Source Code version that alters or restricts the applicable version of this License or the recipients' rights hereunder. However, You may include an additional document offering the additional rights described in Section 3.5.

Availability of Source Code

Any Modification which You create or to which You contribute must be made available in Source Code form under the terms of this License either on the same media as an Executable version or via an accepted Electronic Distribution Mechanism to anyone to whom you made an Executable version available; and if made available via Electronic Distribution Mechanism, must remain available for at least twelve (12) months after the date it initially became available, or at least six (6) months after a subsequent version of that particular Modification has been made available to such recipients. You are responsible for ensuring that the Source Code version remains available even if the Electronic Distribution Mechanism is maintained by a third party.

Description of Modifications.

You must cause all Covered Code to which you contribute to contain a file documenting the changes You made to create that Covered Code and the date

of any change. You must include a prominent statement that the Modification is derived, directly or indirectly, from Original Code provided by the Initial Developer and including the name of the Initial Developer in (a) the Source Code, and (b) in any notice in an Executable version or related documentation in which You describe the origin or ownership of the Covered Code.

Intellectual Property Matters

a) **Third Party Claims** If You have knowledge that a party claims an intellectual property right in particular functionality or code (or its utilization under this License), you must include a text file with the source code distribution titled “LEGAL” which describes the claim and the party making the claim in sufficient detail that a recipient will know whom to contact. If you obtain such knowledge after You make Your Modification available as described in Section 3.2, You shall promptly modify the LEGAL file in all copies You make available thereafter and shall take other steps (such as notifying appropriate mailing lists or newsgroups) reasonably calculated to inform those who received the Covered Code that new knowledge has been obtained.

b) **Contributor APIs.** If Your Modification is an application programming interface and You own or control patents which are reasonably necessary to implement that API, you must also include this information in the LEGAL file.

Required Notices

You must duplicate the notice in Exhibit A in each file of the Source Code, and this License in any documentation for the Source Code, where You describe recipients’ rights relating to Covered Code. If You created one or more Modification(s), You may add your name as a Contributor to the notice described in Exhibit A. If it is not possible to put such notice in a particular Source Code file due to its structure, then you must include such notice in a location (such as a relevant directory file) where a user would be likely to look for such a notice. You may choose to offer, and to charge a fee for, warranty, support, indemnity or liability obligations to one or more recipients of Covered Code. However, You may do so only on Your own behalf, and not on behalf of

the Initial Developer or any Contributor. You must make it absolutely clear than any such warranty, support, indemnity or liability obligation is offered by You alone, and You hereby agree to indemnify the Initial Developer and every Contributor for any liability incurred by the Initial Developer or such Contributor as a result of warranty, support, indemnity or liability terms You offer.

Distribution of Executable Versions

You may distribute Covered Code in Executable form only if the requirements of Section 3.1-3.5 have been met for that Covered Code, and if You include a notice stating that the Source Code version of the Covered Code is available under the terms of this License, including a description of how and where You have fulfilled the obligations of Section 3.2. The notice must be conspicuously included in any notice in an Executable version, related documentation or collateral in which You describe recipients' rights relating to the Covered Code. You may distribute the Executable version of Covered Code under a license of Your choice, which may contain terms different from this License, provided that You are in compliance with the terms of this License and that the license for the Executable version does not attempt to limit or alter the recipient's rights in the Source Code version from the rights set forth in this License. If You distribute the Executable version under a different license You must make it absolutely clear that any terms which differ from this License are offered by You alone, not by the Initial Developer or any Contributor. You hereby agree to indemnify the Initial Developer and every Contributor for any liability incurred by the Initial Developer or such Contributor as a result of any such terms You offer.

Larger Works

You may create a Larger Work by combining Covered Code with other code not governed by the terms of this License and distribute the Larger Work as a single product. In such a case, You must make sure the requirements of this License are fulfilled for the Covered Code.

A.7.4 Inability to Comply Due to Statute or Regulation

If it is impossible for You to comply with any of the terms of this License with respect to some or all of the Covered Code due to statute or regulation then You must: (a) comply with the terms of this License to the maximum extent possible; and (b) describe the limitations and the code they affect. Such description must be included in the LEGAL file described in Section 3.4 and must be included with all distributions of the Source Code. Except to the extent prohibited by statute or regulation, such description must be sufficiently detailed for a recipient of ordinary skill to be able to understand it.

A.7.5 Application of this License

This License applies to code to which the Initial Developer has attached the notice in Exhibit A, and to related Covered Code.

A.7.6 Versions of the License

New Versions

Netscape Communications Corporation ("Netscape") may publish revised and-/or new versions of the License from time to time. Each version will be given a distinguishing version number.

Effect of New Versions

Once Covered Code has been published under a particular version of the License, You may always continue to use it under the terms of that version. You may also choose to use such Covered Code under the terms of any subsequent version of the License published by Netscape. No one other than Netscape has the right to modify the terms applicable to Covered Code created under this License.

Derivative Works.

If you create or use a modified version of this License (which you may only do in order to apply it to code which is not already Covered Code governed by this License), you must (a) rename Your license so that the phrases "Mozilla",

"MOZILLAPL", "MOZPL", "Netscape", "NPL" or any confusingly similar phrase do not appear anywhere in your license and (b) otherwise make it clear that your version of the license contains terms which differ from the Mozilla Public License and Netscape Public License. (Filling in the name of the Initial Developer, Original Code or Contributor in the notice described in Exhibit A shall not of themselves be deemed to be modifications of this License.)

A.7.7 DISCLAIMER OF WARRANTY

COVERED CODE IS PROVIDED UNDER THIS LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, WITHOUT LIMITATION, WARRANTIES THAT THE COVERED CODE IS FREE OF DEFECTS, MERCHANTABLE, FIT FOR A PARTICULAR PURPOSE OR NON-INFRINGING. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE COVERED CODE IS WITH YOU. SHOULD ANY COVERED CODE PROVE DEFECTIVE IN ANY RESPECT, YOU (NOT THE INITIAL DEVELOPER OR ANY OTHER CONTRIBUTOR) ASSUME THE COST OF ANY NECESSARY SERVICING, REPAIR OR CORRECTION. THIS DISCLAIMER OF WARRANTY CONSTITUTES AN ESSENTIAL PART OF THIS LICENSE. NO USE OF ANY COVERED CODE IS AUTHORIZED HEREUNDER EXCEPT UNDER THIS DISCLAIMER.

A.7.8 TERMINATION

This License and the rights granted hereunder will terminate automatically if You fail to comply with terms herein and fail to cure such breach within 30 days of becoming aware of the breach. All sublicenses to the Covered Code which are properly granted shall survive any termination of this License. Provisions which, by their nature, must remain in effect beyond the termination of this License shall survive.

A.7.9 LIMITATION OF LIABILITY

UNDER NO CIRCUMSTANCES AND UNDER NO LEGAL THEORY, WHETHER TORT (INCLUDING NEGLIGENCE), CONTRACT, OR OTHER-

WISE, SHALL THE INITIAL DEVELOPER, ANY OTHER CONTRIBUTOR, OR ANY DISTRIBUTOR OF COVERED CODE, OR ANY SUPPLIER OF ANY OF SUCH PARTIES, BE LIABLE TO YOU OR ANY OTHER PERSON FOR ANY INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES OF ANY CHARACTER INCLUDING, WITHOUT LIMITATION, DAMAGES FOR LOSS OF GOODWILL, WORK STOPPAGE, COMPUTER FAILURE OR MALFUNCTION, OR ANY AND ALL OTHER COMMERCIAL DAMAGES OR LOSSES, EVEN IF SUCH PARTY SHALL HAVE BEEN INFORMED OF THE POSSIBILITY OF SUCH DAMAGES. THIS LIMITATION OF LIABILITY SHALL NOT APPLY TO LIABILITY FOR DEATH OR PERSONAL INJURY RESULTING FROM SUCH PARTY'S NEGLIGENCE TO THE EXTENT APPLICABLE LAW PROHIBITS SUCH LIMITATION. SOME JURISDICTIONS DO NOT ALLOW THE EXCLUSION OR LIMITATION OF INCIDENTAL OR CONSEQUENTIAL DAMAGES, SO THAT EXCLUSION AND LIMITATION MAY NOT APPLY TO YOU.

A.7.10 U.S. GOVERNMENT END USERS

The Covered Code is a “commercial item,” as that term is defined in 48 C.F.R. 2.101 (Oct. 1995), consisting of “commercial computer software” and “commercial computer software documentation,” as such terms are used in 48 C.F.R. 12.212 (Sept. 1995). Consistent with 48 C.F.R. 12.212 and 48 C.F.R. 227.7202-1 through 227.7202-4 (June 1995), all U.S. Government End Users acquire Covered Code with only those rights set forth herein.

A.7.11 MISCELLANEOUS

This License represents the complete agreement concerning subject matter hereof. If any provision of this License is held to be unenforceable, such provision shall be reformed only to the extent necessary to make it enforceable. This License shall be governed by California law provisions (except to the extent applicable law, if any, provides otherwise), excluding its conflict-of-law provisions. With respect to disputes in which at least one party is a citizen of, or an entity chartered or registered to do business in, the United States of America: (a) unless otherwise agreed in writing, all disputes relating to this

License (excepting any dispute relating to intellectual property rights) shall be subject to final and binding arbitration, with the losing party paying all costs of arbitration; (b) any arbitration relating to this Agreement shall be held in Santa Clara County, California, under the auspices of JAMS/EndDispute; and (c) any litigation relating to this Agreement shall be subject to the jurisdiction of the Federal Courts of the Northern District of California, with venue lying in Santa Clara County, California, with the losing party responsible for costs, including without limitation, court costs and reasonable attorneys fees and expenses. The application of the United Nations Convention on Contracts for the International Sale of Goods is expressly excluded. Any law or regulation which provides that the language of a contract shall be construed against the drafter shall not apply to this License.

A.7.12 RESPONSIBILITY FOR CLAIMS

Except in cases where another Contributor has failed to comply with Section 3.4, You are responsible for damages arising, directly or indirectly, out of Your utilization of rights under this License, based on the number of copies of Covered Code you made available, the revenues you received from utilizing such rights, and other relevant factors. You agree to work with affected parties to distribute responsibility on an equitable basis.

A.7.13 EXHIBIT A

“The contents of this file are subject to the Mozilla Public License Version 1.0 (the “License”); you may not use this file except in compliance with the License. You may obtain a copy of the License at http://www.mozilla.org-/MPL/

Software distributed under the License is distributed on an “AS IS” basis, WITHOUT WARRANTY OF ANY KIND, either express or implied. See the License for the specific language governing rights and limitations under the License.

The Original Code is ______________________________________.

The Initial Developer of the Original Code is ______________________-__. Portions created by ______________________ are Copyright (C) ______ ________________________. All Rights Reserved.

Contributor(s): ______________________________."

A.8 The QPL

The intent of this license is to establish freedom to share and change the software regulated by this license under the open source model.

This license applies to any software containing a notice placed by the copyright holder saying that it may be distributed under the terms of the Q Public License version 1.0. Such software is herein referred to as the Software. This license covers modification and distribution of the Software, use of third-party application programs based on the Software, and development of free software which uses the Software.

A.8.1 Granted Rights

1. You are granted the non-exclusive rights set forth in this license provided you agree to and comply with any and all conditions in this license. Whole or partial distribution of the Software, or software items that link with the Software, in any form signifies acceptance of this license.

2. You may copy and distribute the Software in unmodified form provided that the entire package, including - but not restricted to - copyright, trademark notices and disclaimers, as released by the initial developer of the Software, is distributed.

3. You may make modifications to the Software and distribute your modifications, in a form that is separate from the Software, such as patches. The following restrictions apply to modifications:

   a) Modifications must not alter or remove any copyright notices in the Software.

   b) When modifications to the Software are released under this license, a non-exclusive royalty-free right is granted to the initial developer of the Software to distribute your modification in future versions of the Software provided such versions remain available under these terms in addition to any other license(s) of the initial developer.

4. You may distribute machine-executable forms of the Software or machine-executable forms of modified versions of the Software, provided that you meet these restrictions:

   a) You must include this license document in the distribution.

   b) You must ensure that all recipients of the machine-executable forms are also able to receive the complete machine-readable source code to the distributed Software, including all modifications, without any charge beyond the costs of data transfer, and place prominent notices in the distribution explaining this.

   c) You must ensure that all modifications included in the machine-executable forms are available under the terms of this license.

5. You may use the original or modified versions of the Software to compile, link and run application programs legally developed by you or by others.

6. You may develop application programs, reusable components and other software items that link with the original or modified versions of the Software. These items, when distributed, are subject to the following requirements:

   a) You must ensure that all recipients of machine-executable forms of these items are also able to receive and use the complete machine-readable source code to the items without any charge beyond the costs of data transfer.

   b) You must explicitly license all recipients of your items to use and re-distribute original and modified versions of the items in both machine-executable and source code forms. The recipients must be able to do so without any charges whatsoever, and they must be able to re-distribute to anyone they choose.

   c) If the items are not available to the general public, and the initial developer of the Software requests a copy of the items, then you must supply one.

A.8.2 Limitations of Liability

In no event shall the initial developers or copyright holders be liable for any damages whatsoever, including - but not restricted to - lost revenue or profits or other direct, indirect, special, incidental or consequential damages, even if they have been advised of the possibility of such damages, except to the extent invariable law, if any, provides otherwise.

A.8.3 No Warranty

The Software and this license document are provided AS IS with NO WARRANTY OF ANY KIND, INCLUDING THE WARRANTY OF DESIGN, MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

A.8.4 Choice of Law

This license is governed by the Laws of Norway. Disputes shall be settled by Oslo City Court.

A.9 IBM PUBLIC LICENSE VERSION 1.0 - JIKES COMPILER

THE ACCOMPANYING PROGRAM IS PROVIDED UNDER THE TERMS OF THIS IBM PUBLIC LICENSE (“AGREEMENT”). ANY USE, REPRODUCTION OR DISTRIBUTION OF THE PROGRAM CONSTITUTES RECIPIENT’S ACCEPTANCE OF THIS AGREEMENT.

A.9.1 DEFINITIONS

“Contribution” means:

in the case of International Business Machines Corporation (“IBM”), the Original Program, and in the case of each Contributor, changes to the Program, and additions to the Program; where such changes and/or additions to the Program originate from and are distributed by that particular Contributor. A Contribution ’originates’ from a Contributor if it was added to the Program

by such Contributor itself or anyone acting on such Contributor's behalf. Contributions do not include additions to the Program which: (i) are separate modules of software distributed in conjunction with the Program under their own license agreement, and (ii) are not derivative works of the Program.

"Contributor" means IBM and any other entity that distributes the Program.

"Licensed Patents" mean patent claims licensable by a Contributor which are necessarily infringed by the use or sale of its Contribution alone or when combined with the Program.

"Original Program" means the original version of the software accompanying this Agreement as released by IBM, including source code, object code and documentation, if any.

"Program" means the Original Program and Contributions.

"Recipient" means anyone who receives the Program under this Agreement, including all Contributors.

A.9.2 GRANT OF RIGHTS

Subject to the terms of this Agreement, each Contributor hereby grants Recipient a non-exclusive, worldwide, royalty-free copyright license to reproduce, prepare derivative works of, publicly display, publicly perform, distribute and sublicense the Contribution of such Contributor, if any, and such derivative works, in source code and object code form.

Subject to the terms of this Agreement, each Contributor hereby grants Recipient a non-exclusive, worldwide, royalty-free patent license under Licensed Patents to make, use, sell, offer to sell, import and otherwise transfer the Contribution of such Contributor, if any, in source code and object code form. This patent license shall apply to the combination of the Contribution and the Program if, at the time the Contribution is added by the Contributor, such addition of the Contribution causes such combination to be covered by the Licensed Patents. The patent license shall not apply to any other combinations which include the Contribution. No hardware per se is licensed hereunder.

Recipient understands that although each Contributor grants the licenses to its Contributions set forth herein, no assurances are provided by any Contributor that the Program does not infringe the patent or other intellectual pro-

perty rights of any other entity. Each Contributor disclaims any liability to Recipient for claims brought by any other entity based on infringement of intellectual property rights or otherwise. As a condition to exercising the rights and licenses granted hereunder, each Recipient hereby assumes sole responsibility to secure any other intellectual property rights needed, if any. For example, if a third party patent license is required to allow Recipient to distribute the Program, it is Recipient's responsibility to acquire that license before distributing the Program.

Each Contributor represents that to its knowledge it has sufficient copyright rights in its Contribution, if any, to grant the copyright license set forth in this Agreement.

A.9.3 REQUIREMENTS

A Contributor may choose to distribute the Program in object code form under its own license agreement, provided that:

it complies with the terms and conditions of this Agreement; and

its license agreement:

- effectively disclaims on behalf of all Contributors all warranties and conditions, express and implied, including warranties or conditions of title and non-infringement, and implied warranties or conditions of merchantability and fitness for a particular purpose;
- effectively excludes on behalf of all Contributors all liability for damages, including direct, indirect, special, incidental and consequential damages, such as lost profits;
- states that any provisions which differ from this Agreement are offered by that Contributor alone and not by any other party; and
- states that source code for the Program is available from such Contributor, and informs licensees how to obtain it in a reasonable manner on or through a medium customarily used for software exchange.

When the Program is made available in source code form:

it must be made available under this Agreement; and a copy of this Agreement must be included with each copy of the Program.

Each Contributor must include the following in a conspicuous location in the Program:

> Copyright (C) 1996, 1999 International Business Machines Corporation and others. All Rights Reserved.

In addition, each Contributor must identify itself as the originator of its Contribution, if any, in a manner that reasonably allows subsequent Recipients to identify the originator of the Contribution.

A.9.4 COMMERCIAL DISTRIBUTION

Commercial distributors of software may accept certain responsibilities with respect to end users, business partners and the like. While this license is intended to facilitate the commercial use of the Program, the Contributor who includes the Program in a commercial product offering should do so in a manner which does not create potential liability for other Contributors. Therefore, if a Contributor includes the Program in a commercial product offering, such Contributor (“Commercial Contributor”) hereby agrees to defend and indemnify every other Contributor (“Indemnified Contributor”) against any losses, damages and costs (collectively “Losses”) arising from claims, lawsuits and other legal actions brought by a third party against the Indemnified Contributor to the extent caused by the acts or omissions of such Commercial Contributor in connection with its distribution of the Program in a commercial product offering. The obligations in this section do not apply to any claims or Losses relating to any actual or alleged intellectual property infringement. In order to qualify, an Indemnified Contributor must: a) promptly notify the Commercial Contributor in writing of such claim, and b) allow the Commercial Contributor to control, and cooperate with the Commercial Contributor in, the defense and any related settlement negotiations. The Indemnified Contributor may participate in any such claim at its own expense.

For example, a Contributor might include the Program in a commercial product offering, Product X. That Contributor is then a Commercial Contributor. If that Commercial Contributor then makes performance claims, or offers warranties related to Product X, those performance claims and warranties are such Commercial Contributor’s responsibility alone. Under this section, the Commercial Contributor would have to defend claims against the

other Contributors related to those performance claims and warranties, and if a court requires any other Contributor to pay any damages as a result, the Commercial Contributor must pay those damages.

A.9.5 NO WARRANTY

EXCEPT AS EXPRESSLY SET FORTH IN THIS AGREEMENT, THE PROGRAM IS PROVIDED ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED INCLUDING, WITHOUT LIMITATION, ANY WARRANTIES OR CONDITIONS OF TITLE, NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Each Recipient is solely responsible for determining the appropriateness of using and distributing the Program and assumes all risks associated with its exercise of rights under this Agreement, including but not limited to the risks and costs of program errors, compliance with applicable laws, damage to or loss of data, programs or equipment, and unavailability or interruption of operations.

A.9.6 DISCLAIMER OF LIABILITY

EXCEPT AS EXPRESSLY SET FORTH IN THIS AGREEMENT, NEITHER RECIPIENT NOR ANY CONTRIBUTORS SHALL HAVE ANY LIABILITY FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING WITHOUT LIMITATION LOST PROFITS), HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OR DISTRIBUTION OF THE PROGRAM OR THE EXERCISE OF ANY RIGHTS GRANTED HEREUNDER, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

A.9.7 GENERAL

If any provision of this Agreement is invalid or unenforceable under applicable law, it shall not affect the validity or enforceability of the remainder of the terms of this Agreement, and without further action by the parties hereto,

such provision shall be reformed to the minimum extent necessary to make such provision valid and enforceable.

If Recipient institutes patent litigation against a Contributor with respect to a patent applicable to software (including a cross-claim or counterclaim in a lawsuit), then any patent licenses granted by that Contributor to such Recipient under this Agreement shall terminate as of the date such litigation is filed. In addition, if Recipient institutes patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Program itself (excluding combinations of the Program with other software or hardware) infringes such Recipient's patent(s), then such Recipient's rights granted under Section 2(b) shall terminate as of the date such litigation is filed. All Recipient's rights under this Agreement shall terminate if it fails to comply with any of the material terms or conditions of this Agreement and does not cure such failure in a reasonable period of time after becoming aware of such noncompliance. If all Recipient's rights under this Agreement terminate, Recipient agrees to cease use and distribution of the Program as soon as reasonably practicable. However, Recipient's obligations under this Agreement and any licenses granted by Recipient relating to the Program shall continue and survive.

IBM may publish new versions (including revisions) of this Agreement from time to time. Each new version of the Agreement will be given a distinguishing version number. The Program (including Contributions) may always be distributed subject to the version of the Agreement under which it was received. In addition, after a new version of the Agreement is published, Contributor may elect to distribute the Program (including its Contributions) under the new version. No one other than IBM has the right to modify this Agreement. Except as expressly stated in Sections 2(a) and 2(b) above, Recipient receives no rights or licenses to the intellectual property of any Contributor under this Agreement, whether expressly, by implication, estoppel or otherwise. All rights in the Program not expressly granted under this Agreement are reserved.

This Agreement is governed by the laws of the State of New York and the intellectual property laws of the United States of America. No party to this Agreement will bring a legal action under this Agreement more than one year after the cause of action arose. Each party waives its rights to a jury trial in any resulting litigation.

B Il patrimonio Open Source

Questo capitolo presenta i progetti Open Source più noti al fine di dare una tangibile testimonianza della validità del metodo di sviluppo al di fuori del riuscito canone di Linux. I progetti riportati sono solo una piccola parte di quelli esistenti ma è un campione altamente rappresentativo. Sono inclusi linguaggi di scripting, software di gestione protocolli per Internet e l'onnipresente GNU. Si consideri che solo GNU offre centinaia di programmi: dai compilatori, alla grafica, agli editing. . . ecc., l'intero ambiente UNIX arricchito di intere suite.

Tutto ciò è stato creato grazie alla volontà di pochi ed è liberamente disponibile per tutti. Se il fenomeno Open Source si espanderà anche presso l'impresa potremo finalmente contare su un patrimonio inestinguibile e in continuo progresso di software liberamente disponibile.

B.1 GNU

Del progetto GNU si è già ampiamente discusso nel corso della tesi. Qui si vogliono ricordare le linee essenziali e ricordare ancora una volta la sua importanza per lo sviluppo del Free Software.

Nel 1983, quando AT&T pose sotto licenza commerciale il sistema Unix, Richard Stallman, ricercatore del Massachusets Institute of Technology (MIT), lanciò un progetto per costruire uno Unix alternativo totalmente libero chiamato GNU. Il progetto di Stallman non si limitava al sistema operativo ma all'intero ambiente: compilatori, editor, giochi, utilità, applicativi.

Stallman creò la "Free Software Foundation" (FSF), un'organizzazione per promuovere l'idea delle piena disponibilità e distribuzione del codice sorgente di tutti i programmi. La licenza promossa dalla FSF è GNU Public License che stabilisce i termini con cui rendere disponibile il codice sorgente e specifica che qualsiasi programma che includa codice coperto dalla GPL deve essere esso stesso reso disponibile sotto i termini della GPL.

Centinaia di programmatori si associarono al progetto creando versioni Free Software dei maggiori programmi Unix. Il progetto che avrebbe dovuto portare alla costruzione del nucleo del sistema operativo incontrò serie difficoltà e tutt'oggi non è ancora utilizzabile. Tuttavia il ricco patrimonio della FSF è parte integrante delle distribuzioni Linux.

Molti dei prodotti GNU sono così potenti da essere diventati parte integrante di tutti i sistemi Unix. In particolare GCC divenne il compilatore C dominante e GNU EMACS l'editor di programmazione più usato.

La GPL permette la vendita del software provvisto del codice sorgente, anche se stabilisce che il prezzo del software deve essere relativo al supporto con cui viene distribuito. La licenza GPL, stabilendo l'obbligatorietà della distribuzione del codice sorgente per ogni lavoro basato su codice coperto da GPL, limitò l'uso in ambito commerciale del compilatore GCC. Una licenza più permissiva (LGPL) fu rilasciata per ovviare al problema, ma l'approccio dogmatico della FSF riguardo il Free Software portò molti suoi sostenitori a creare licenze più vicine al mondo commerciale.

La lista dei programmi creati in seno al progetto GNU occuperebbe un'intera tesi, per avere un elenco completo si visiti il sito della FSF (www.fsf.org).

B.2 BSD

Quando Unix veniva sviluppato in seno ai laboratori AT&T Bell Labs non si poteva parlare di software libero nei termini correnti, ma il codice sorgente di Unix era disponibile, per una somma simbolica, ai laboratori universitari. La possibilità di lavorare direttamente sul sistema Unix portò a un'esplosione di creatività incentivata dalla libera distribuzione dei programmi tra i ricercatori.

La principale risorsa di sviluppo per Unix al di fuori dei Bell Labs fu la "University of California at Berkeley". Berkeley produsse programmi che divennero parte integrante del sistema Unix. Il più importante sviluppo fu l'implementazione del protocollo TCP/IP, la diffusione del quale costituì le fondamenta di Internet.

Berkeley Unix è conosciuto col nome di "Berkeley Standard Distribution" (BSD). Il progetto BSD non è più parte dei programmi di sviluppo dell'università di Berkeley ma il software continua a essere mantenuto dal "FreeBSD Project". È da ricordare che dopo la decisione di AT&T di commercializzare Unix lo sviluppo di UNIX stagnò per molto tempo. Molti tentativi di crea-

re una distribuzione cononica commerciale di Unix fallirono lasciando così spazio alla nascente Microsoft per imporre il proprio DOS sulle macchine di fascia più bassa.

Il patrimonio del progetto Unix Berkeley venne raccolto e gestito dal suo direttore, Bill Joy, fondando la Sun. La Sun fu in grado di imporsi sul mercato grazie al fatto di poter disporre di un sistema operativo già collaudato. Compagnie come Cisco, Frontier, NetManage e altre basarono i loro iniziali prodotti sull'implementazione del protocollo TCP/IP di Berkeley.

Il progetto FreeBSD partì nel 1993 guidato da Nate Williams, Rod Grimes e Jordan K. Hubbard. Il primo passo fu il rilascio di una fix per il 386BSD, una versione di BSD per piattaforme Intel. Il progetto fu supportato dalla "Walnut Creek CDROM", compagnia interessata nella distribuzione del sistema operativo. Nel Dicembre del 1993 venne rilasciata la prima distribuzione di FreeBSD 1.0, basata sul 4.3BSD-Lite e su componenti forniti dal 386BSD e dalla Free Software Foundation. Il FreeBSD 1.1 fu rilasciato nel Maggio 1994.

Dopo una battaglia legale tra Novell (neoproprietaria del sistema Unix) e UC Berkeley su chi possedesse cosa il progetto FreeBSD dovette quasi ripartire da capo per sostituire le parti di codice riconosciute come proprietà di Novell. Nel Gennaio del 1995 venne rilasciato FreeBSD 2.0. La diffusione del sistema operativo fu un successo e venne deciso un aggiornamento nel Giugno successivo. Di versione in versione si è giunti nel 1999 al FreeBSD 3.0.

B.3 DNS e BIND

Il nome BIND (Berkeley Internet Name Domain) non è molto conosciuto al di fuori degli addetti ai lavori in Internet, ma chiunque utilizza il servizio che BIND rende possibile: il Domain Name Server (DNS). La rete di server BIND rende possibile che il DNS traduca gli indirizzi Internet come 207.25.98.191 nella forma che solitamente usiamo: polito.it, reply.it, ecc.

Originariamente sviluppato da Paul Mockapetris nel 1984, DNS è attualmente mantenuto da Paul Vixie sotto l'egida dell'Internet Software Consortium.

Il DNS, insieme al protocollo TCP/IP, costituiscono le fondamenta del-

l'intera industria basata su Internet: oltre 29 milioni di siti hanno registrato il nome di dominio.

DNS nacque con l'esigenza di gestire la diffusione mondiale di Internet. Negli anni '70, quando Arpanet era una piccola comunità di circa un centinaio di host, un singolo file, HOSTS.TXT, conteneva tutte le informazioni necessarie alla gestione degli indirizzi: una mappa nome-indirizzo per ogni host connesso a Arpanet.

Il file era mantenuto dal Network Information Center (detto "The NIC") dello Stanford Research Institute (SRI) e distribuito da un singolo host: SRI-NIC.

Gli amministratori di Arpanet erano soliti mandare un'e-mail con le modifiche al NIC e periodicamente scaricavano con Ftp il file host.txt aggiornato che veniva compilato sulle proprie macchine una o due volte alla settimana. Con il crescere di Arpanet, più nel numero di host che nelle capacità di rete, questo metodo divenne insostenibile, il traffico generato dal processo di aggiornamento degli indirizzi divenne troppo oneroso. Quando Arpanet adotto il protocollo TCP/IP, la popolazione della rete esplose e si rilevarono tre grossi problemi con HOST.TXT:

- La gestione di SRI-NIC in termini di traffico di rete e di carico del processore divenne molto pesante.

- Due host non potevano avere lo stesso nome su HOST.TXT. Benché il NIC potesse assegnare gli indirizzi in modo da garantirne l'unicità non aveva nessuna autorità sui nomi degli host. Non c'era nulla che potesse prevenire il fatto che qualcuno aggiungesse un host con un nome già utilizzato e danneggiare il servizio.

- Il mantenimento della consistenza del file HOSTS.TXT durante l'espansione della rete divenne difficile. Tenere traccia degli indirizzi e dei nomi divenne un lavoro a tempo pieno e con risultati scadenti.

Il problema essenziale era che la gestione basata sul file HOST.TXT non era per nulla scalabile. Venne cercata una soluzione che permettesse la massima scalabilità e il massimo automatismo, avrebbe dovuto permettere una gestione locale dei dati e una loro disponibilità globale. La decentralizzazione del sistema avrebbe eliminato il collo di bottiglia causato dal server unico

per la gestione degli indirizzi e alleggerito il traffico su rete. La gestione locale dei dati avrebbe mantenuto l'aggiornamento dei dati più facile da gestire. Fu deciso di adottare una struttura gerarchica sui nomi per garantire l'unicità dei nomi.

Paul Mockapetris, dell'Information Sciences Institute dell'USC, fu nominato responsabile della progettazione della nuova architettura dei nomi. Nel 1984 rilasciò gli RFC 882 e 883, che descrivono il Domain Name System o DNS. Quei RFC furono aggiornati nelle versioni 1034 e 1035, che ancora oggi contengono le specifiche del DNS, e poi estesi nei documenti 1535, 1536 e 1537 per coprire problemi di sicurezza e di amministrazione.

La prima implementazione del DNS fu chiamata JEEVES e fu scritta da Paul Mockapetris stesso. Un'implementazione posteriore fu BIND, scritta da Kevin Dunlap per la versione BSD UNIX 4.3. BIND è ora mantenuto da Paul Vixie.

BIND significa Berkeley Internet Name Domain ed è di gran lunga la usata implementazione del DNS, utilizzata dai sistemi operativi più diffusi sul mercato.

B.4 Perl

Originariamente sviluppato nel 1986 da Larry Wall, Perl (Practical Extraction and Report Language) è divenuto il linguaggio scelto per l'amministrazione di rete e di sistema e per la programmazione CGI. Perl è stato indicato come "la colla universale di Internet" per il modo in cui collega insieme i processi più disparati. Grandi siti come Netscape, Yahoo, CNET, Amazon e Excite fanno un intenso uso di Perl per la gestione e per la fornitura di servizi agli utenti.

Perl è ora mantenuto da un gruppo di circa 100 sviluppatori che si tengono in contatto mediante la perl5porters mailing list. Larry Wall mantiene un controllo "artistico" sulla forma del linguaggio, è da considerare comunque il fatto che grazie a un ben definito meccanismo di estensione sono stati sviluppati circa 600 moduli aggiuntivi sviluppati da programmatori indipendenti.

Perl è incluso nei sistemi Unix così come nei Microsoft's NT Resource Pack. È il linguaggio primario usato per creare i servizi su Internet, specialmente i siti a contenuto dinamico.

Si ritiene che sia utilizzato da circa 500000 programmatori e che decine di milioni di utenti ne utilizzino i servigi.

B.5 Python

Python è un ulteriore esempio di linguaggio nato per scopi limitati e particolari e poi esteso per coprire le necessità degli sviluppatori di siti Web. Guido van Rossum, di Amsterdam, è lo sviluppatore di Python, da lui concepito come linguaggio avanzato di scripting, un nuovo e migliore Perl. Sarebbe difficile dire qual'è il linguaggio migliore tra Perl e Python, entrambi hanno le loro schiere di seguaci.

La "Python Software Association" fu fondata sotto l'egida della "Corporation for National Research Initiatives" per mantenere lo sviluppo di Python.

Python nacque nel 1990 e deve il suo nome a una serie televisiva della BBC (la "Monty Python's Flying Circus"). Venne impiegato dapprima sul sistema operativo Amoeba come linguaggio di scripting avanzato e successivamente esteso su sistemi Unix e PC. Oggi è usato per innumerevoli scopi: lo sviluppo di strumenti GUI, WWW scripting, televisione interattiva, rapid programming development, personalizzazione di librerie C++ e altro ancora. Dalla sua comparsa nel 1991 come programma di pubblico dominio nel 1991 ha continuato ad attrarre programmatori, un news gli è interamente dedicato: comp.lang.python, dal 1994.

Benché sia di pubblico dominio Python è un sistema ben supportato, grazie all'attività del suo inventore e della comunità che lo segue. Python non ha limiti di utilizzo o distribuzione e viene distribuito accompagnato di codice sorgente, un debugger, un profiler, interfacce per dispositivi esterni più strumenti per l'aggiunta di ulteriori componenti. Altri moduli che ne estendono la gamma di applicazioni vengono forniti da distribuzioni indipendenti.

B.6 Sendmail

Originariamente sviluppato da Eric Allman nel 1981, Sendmail è l'agente di trasporto della posta più usato su Internet, con un mercato stimato intorno al 75-80

Nel Novembre 1997 Eric Allman e Greg Olson fondarono la Sendmail, Inc. per distribuire una versione commerciale di Sendmail. La compagnia continua a seguire lo sviluppo della versione freeware garantendo la distribuzione del codice sorgente e il diritto di modifica. Il prodotto commerciale si focalizzerà sui tool di configurazione e sull'integrazione con la gestione di siti www. Sendmail è usato dalla maggioranza degli Internet Service Provider ed è distribuito con praticamente tutte le distribuzioni Linux (Sun, HP, IBM, DEC, SGI, SCO, ecc.)

Il programma Sendmail venne sviluppato da Eric Allman quando era studente e staff member alla University of California at Berkeley. A quel tempo un computer (Ingres) del campus era connessa a ARPAnet, un altro computer (Ernie CoVax) era connesso a UUCP, queste macchine (più altre) erano connesse tra di loro mediante una rete a basso costo interna all'Università, la BerkNet. La posta poteva girare internamente ad ARPAnet, ad UUCP e BerkNet, ma non c'era modo che le mail venissero inoltrate da una rete all'altra.

L'improvvisa crescita del numero di protocollo e la conseguente nascita di reti diverse spinsero Allman a scrivere Delivermail, il precursore di Sendmail. Delivermail venne distribuito nel 1979 con le versioni BSD UNIX 4.0 e 4.1. Sfortunatamente Delivermail non era abbastanza flessibile per seguire il continuo mutare delle condizioni di routing, il suo più grande limite era costituito dal fatto che la sua configurazione doveva essere compilata all'interno del programma.

Nel 1980, ARPAnet passò ad utilizzare il protocollo TCP. Questa modifica consentiva di passare da un possibile numero massimo di 256 host a un bilione. Un altro cambiamento era costituito dal passare da un indirizzamento "piatto" ad un indirizzamento "gerarchico". Prima di questi cambiamenti la posta veniva inoltrata utilizzando il protocollo FTP, ora era disponibile il protocollo SMTP (Simple Mail Transport Protocoll) anche se non in versione definitiva.

Per rispondere a questi e ad altri mutamenti Allman riscrisse Delivermail e lo ribattezzò Sendmail. Per assicurare che i messaggi fossero trasferiti tra reti diverse Allman adottò un approccio liberale modificando gli header delle mail secondo le necessità di indirizzamento. Per esempio il protocollo UUCP non prevedeva headers per le mail, Sendmail gestiva i pacchetti UUCP ricavando gli headers dalle informazioni del protocollo.

La prima versione di Sendmail fu distribuita con la versione BSD 4.1c, la

prima versione di Unix a poter usare il protocollo TCP/IP. Dalla prima versione ad oggi Allman ha continuato (con una lunga pausa tra il 1982 al 1990) a migliorare ed estendere Sendmail, prima alla UC Berkeley, poi alla Britton Lee, poi ancora alla UC Berkeley, e ora con InReference Inc. La versione corrente di Sendmail è la 8.x (conosciuta come V8). V8 è una riscrittura quasi completa dell'originario Sendmail che include molte correzioni e significative estensioni.

È da notare che furono in circolazione molte versioni di Sendmail sviluppate da università (Università di Linkoping (Svezia), Northern Illinois University, The University of Illinois) o da produttori commerciali (Digital Equipment Corporation, Sun Microsystem) ciascuna delle quali con pregi e difetti. Dal 1994 Allman si occupa dell'integrazione di queste versioni in un'unico prodotto, la versione V8.8 è ora la versione "ufficiale" di Sendmail.

B.7 Tcl/Tk

John Ousterhout, il creatore di Tcl/Tc, incominciò a lavorare su Tcl come professore alla University of California at Berkeley. Poi si spostò in Sun Microsystems, società interessata nello sviluppo di SunScript, un linguaggio commerciale di scripting. Ora Oustehout si è messo in proprio fondando una nuova società, la Scriptics, per continuare lo sviluppo freeware di Tcl/Tk sfruttando le opportunità commerciali legate agli strumenti di sviluppo Tcl e alla consulenza.

Come Perl e Python, Tcl è un linguaggio di scripting la cui sigla significa "Tool Command Language". Fin dal principio Tcl fu esteso per supportare le applicazioni grafiche tramite Tk, originariamente uno strumento di supporto all'interfaccia grafica di X-Windows. Ora Tcl/Tk supporta anche le interfacce Windows e Mac.

Tcl/Tk è molto diffuso, si contano fino a un milione di attivi sviluppatori. La Sun microsystems conta circa 12000 downloads a settimana così distribuiti a livello di piattaforma: Windows 65

Dei tre linguaggi di scripting citati il Tcl/Tk sembra essere quello di maggiore successo nel mondo Windows. La sua facilità nello sviluppo di applicazioni grafiche trova proseliti provenienti dal mondo di Visual Basic.

Oltre alla facilità d'uso, Tcl fornisce la disponibilità di un ricco insieme di estensioni che permettono agli sviluppatori di risolvere ogni tipo di necessità.

Tcl nacque nel 1988 come linguaggio da usarsi supportato da componenti in C o C++, ma i programmatori incominciarono ad usarlo in una forma standalone portandolo agli attuali livelli.

Tcl (pron. “tickle”) nacque come supporto ai tool di progettazione di circuiti. Il gruppo impegnato in questa attività alla University of California at Berkeley spendeva troppo tempo nell’implementare funzionalità tramite comandi inadeguati. Venne deciso di elaborare un generico linguaggio di comando che avrebbe potuto essere facilmente connesso ad un’applicazione e facilmente esteso con comandi specifici per l’applicazione stessa. In più, essendo la base del linguaggio comune a tutti i moduli, non sarebbe stato necessario imparare una nuova sintassi per ogni applicativo.

Tk (pron. “tee-kay”) fu creato sulla scia di Tcl, Ousterhout usò il nucleo del linguaggio Tcl per inserire delle funzionalità grafiche come insieme di comandi. Scrisse un interprete standalone (wish, “windowing shell”), fornendo così la possibilità di sviluppare applicazioni grafiche a livello di script.

Allo stesso tempo altri incominciarono a scrivere le proprie estensioni Tcl. Due delle prime furono Expect e Extended Tcl (TclX), subito adottate dalla maggior parte dei programmatori Tcl e dimostrarono la validità dell’idea di costruire estensioni basate su un nucleo comune. La potenzialità di Tcl era costituita dal fatto di poter scrivere estensioni senza preoccuparsi di mantenere una congruità progettuale con il linguaggio stesso. L’integrazione dei diversi moduli è molto semplice e consente di estendere virtualmente all’infinito Tcl. Inoltre la base sintattica rimane la stessa per tutte le nuove funzionalità, questo ne facilita l’uso, permettendo di concentrarsi sulle funzionalità piuttosto che nel loro apprendimento. Tcl crebbe molto rapidamente spinto dalla diffusione di wish e di Expect (che è diventato parte integrante delle distribuzioni per amministratori di sistema), sull’onda del successo molti programmatori distribuirono le proprie estensioni, molte delle quali a livello commerciale e adatte a scopi mission-critical.

B.8 PHP

Il PHP è un linguaggio server-side scripting per creare pagine dinamiche per il Web, come JSP, Asp e Coldfusion. Comunque il PHP è Open Source e cross-platform, gira su Windows NT e una varietà di versioni Unix. È potente e molto versatile. Si può usare per accedere a database come MySql, Ora-

cle, Sybase, mSQL, Postgres e a qualsiasi database su NT usando ODBC per creare pagine dinamiche sul web. Oggi il PHP è usato in più di un milione di siti Web ed è in continua crescita.

Fu sviluppato da Rasmus Lerdof a partire dal 1995 partendo da una semplice procedura che avrebbe permesso di scrivere codice C da richiamare dall'interno della pagina web contenente speciali tag. La tecnica si diffuse presso altri sviluppatori web e incominciarono a formarsi le basi per un nuovo linguaggio: vennero aggiunte strutture essenziali quali *if(), for(), while()*. Più ovviamente le variabili. L'impulso iniziale fu molto forte, oggi il PHP è arrivato alla terza versione ed è in cantire la quarta in grado di supportare la programmazione orientata agli oggetti.

PHP è molto indicato per lo sviluppo rapido di HTML dinamico per siti medio piccoli, meno indicatto nel caso di siti molto grossi che necessitano una progettazione ingegneristica dell'applicazione e del contenuto.

B.9 Bibliografia

1. http://www.apogeonline.com/openpress/articoli/freeware2.txt
2. http://www.fsf.org
3. http://www.opensource.org
4. http://www.apogeonline.com/openpress/gnu_gen_pub_lic.html
5. http://www.apogeonline.com/openpress/op_definition.html
6. http://www.fsf.org/philosophy/free-software-for-freedom.html
7. http://www.tuxedo.org/ esr/writings/homesteading/
8. "Open Sources, Voci dalla rivoluzione open source", Apogeo 1999
9. http://www.webreview.com
10. http://www.webreview.com/wr/pub/98/04/10/
11. http://www.tuxedo.org/ esr/writings/cathedral-bazaar
12. http://www.tuxedo.org/ esr/writings/magic-cauldron/

Catalogo Apogeo – luglio 2002



Flash Xbox
di Viscardi Rosario
Pagine: 240
Euro: 7,9

Xbox: oltre i videogiochi, verso il mondo del multimedia, di Internet e del lavoro. Una guida preziosa a tutti i segreti della console Microsoft.



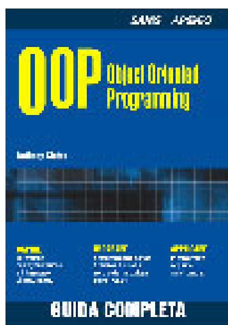
XML Guida Completa
di Devan Shepherd
Pagine: 496
Euro: 34

Dalla sintassi di base fino allo sviluppo di applicazione complete, un tutorial per apprendere i segreti del linguaggio XML in 21 giorni.



Red Hat Linux 7.3 Flash
di Georges Piriou
Pagine: 264
Euro: 7,9

Un manuale tascabile sul sistema operativo libero ideato da Linus Torvalds nella distribuzione Red Hat 7.3.



Object Oriented Programming Guida Completa
di Anthony Sintes
Pagine: 576
Euro: 35,9

Dall'introduzione della programmazione orientata agli oggetti alla presentazione di case study di analisi, progetti e implementazioni.



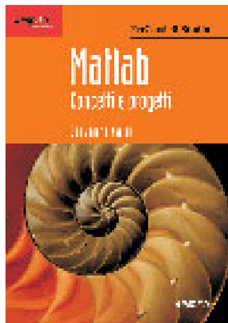
AutoCAD LT 2002 Guida all'uso
di Ralph Grabowski
Pagine: 320
Euro: 25

La guida all'ultima versione di AutoCAD a cura di Ralph Grabowski, autore dei manuali più venduti nel mondo sul più famoso prodotto per il CAD.



Visual Basic .Net Flash
di Marisa Padovani
Pagine: 216
Euro: 7,9

Un piccolo manuale per avere in tasca tutte le informazioni fondamentali su Visual Basic .Net, strutturate secondo brevi lezioni da 10 minuti ciascuna.



Matlab Concetti e progetti
di Naldi Giovanni, Pareschi Lorenzo
Pagine: 360
Euro: 22

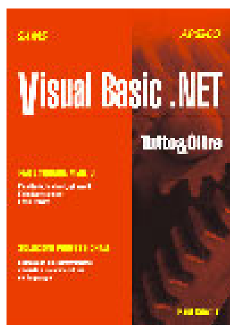
Una introduzione all'uso del software MATLAB come ambiente particolarmente adatto per avvicinarsi al mondo del calcolo scientifico e alle simulazioni numeriche di modelli matematici.



Flash MX Guida all'uso
di M. Mattioli
Pagine: 384
Euro: 25

Come fare dell'animazione la chiave vincente dei propri siti Web. Dalle proprietà di base di Flash ad alcuni elementi di programmazione con ActionScript, il libro affronta tutti gli aspetti più importanti del software.

Catalogo Apogeo – giugno 2002



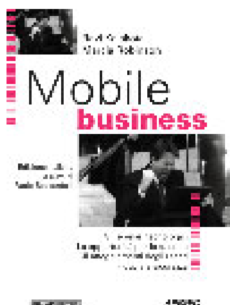
Visual Basic .Net Tutto & Oltre
di Paul Kimmel
Pagine: 600
Euro: 39,9

Come costruire applicazioni distribuite e creare servizi con VB .NET: un manuale indispensabile per il programmatore.



Introduzione alla Macroeconomia
di Kennedy Peter
Pagine: 512
Euro: 30

Un'introduzione, non tecnica ma rigorosa, alla macroeconomia per gli studenti di facoltà universitarie e di corsi post-laurea e per tutti gli interessati a comprendere meglio il funzionamento delle economie in cui viviamo.



Mobile Business
di R. Kalakota, M. Robinson
Pagine: 320
Euro: 23

Diventa sempre più facile e relativamente poco costoso accedere a servizi diversi mediante dispositivi mobili di comunicazione, come cellulari e palmari. Quali sono le prospettive e le opportunità di business?



Strumenti quantitativi per la gestione aziendale
di Waner, Costenoble
Pagine: 360
Euro: 23

La soluzione ideale per l'insegnamento della matematica nelle lauree triennali di Scienze dell'Economia e della Gestione Aziendale.



Fondamenti di telecomunicazioni
di Leon W. Couch
Pagine: 768
Euro: 44

Testo di riferimento per acquisire le competenze fondamentali riguardo ai sistemi di telecomunicazione che ogni professionista del settore dell'Ingegneria e delle Scienze dell'Informazione deve possedere.



[Computer per tutti V edizione](#)
di Enzo Amato
Pagine: 224
Euro: 16,9

Il computer incute ancora timore? Un manuale davvero per tutti per avvicinarsi al mondo della tecnologia e imparare a usarla al meglio.



[JavaScript la guida II edizione](#)
di David Flanagan
Pagine: 816
Euro: 44,9

Dal nucleo del linguaggio fino alla creazione di esempi sofisticati, una guida di riferimento essenziale per il programmatore Javascript.



[Content Management](#)
di Lucchini (a cura di)
Pagine: 420
Euro: 23

Da un master dell’Ateneo Multimediale di Milano, un percorso formativo per quanti devono assolvere alle funzioni di content management.



C# Tutto & Oltre
di Joseph Mayo
Pagine: 672
Euro: 44,9

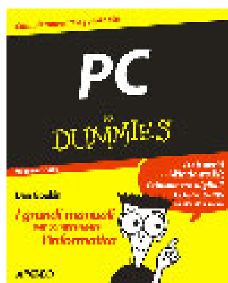
Un testo fondamentale, in grado di mostrare come C# possa essere usato per sviluppare del software come servizio, concetto che sta alla base della suite .NET

Catalogo Apogeo – maggio 2002



ECDL Modulo 4: Foglio elettronico
di Rubini
Pagine: 144
Euro: 9,8

Dal syllabus originale per la patente europea del computer, il modulo 4: Foglio elettronico.



PC For Dummies
di Dan Gookin
Pagine: 320
Euro: 18,9

Un modo facile per avvicinarsi al mondo del computer, con gli ultimi aggiornamenti alla più recente versione di Windows, la XP, i virus, i DVD.



[ECDL Modulo 3: Elaborazione testi]
di Rubini
Pagine: 144
Euro: 9,8

Dal syllabus originale per la patente europea del computer, il modulo 3: Elaborazione testi.



[Thinking in Java]
di Bruce Eckel
Pagine: 768
Euro: 45

Thinking in Java è considerato uno dei testi più autorevoli e al tempo stesso più originali e stimolanti sul linguaggio di programmazione Java.



[ECDL Modulo 5: Basi di dati]
di Rubini
Pagine: 144
Euro: 9,8

Dal syllabus originale per la patente europea del computer, il modulo 5: Basi di dati.



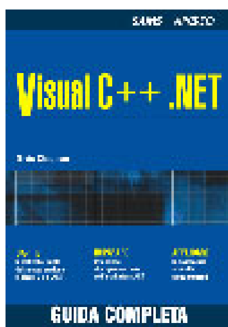
[ECDL Modulo 6: Strumenti di presentazione]
di Rubini
Pagine: 144
Euro: 9,8

Dal syllabus originale per la patente europea del computer, il modulo 6: Strumenti di presentazione.



[AutoCAD 2002 Guida Completa]
di Ralph Grabowski
Pagine: 840
Euro: 44

La terza edizione per il mercato italiano di uno dei più autorevoli testi su AutoCAD, curato da Ralph Grabowski, uno degli autori più noti ed esperti del settore.



[Visual C++ .NET Guida Completa]
di Davis Chapman
Pagine: 704
Euro: 44,9

Progettare finestre d’applicazione, utilizzare i controlli, visualizzare elementi grafici, creare applicazioni SDI e MDI, lavorare con i database e costruire applicazioni multitasking con Visual C++ .NET.



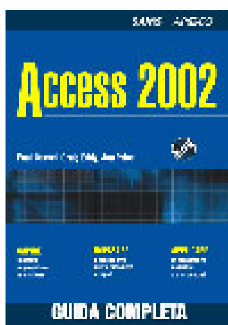
C# Guida Completa
di Bradley L. Jones
Pagine: 672
Euro: 35,9

Il modo migliore per acquisire le tecniche avanzate della programmazione con C# come lo sviluppo di applicazioni in ambiente Windows e la creazione di Web form e Web service.



Internet Explorer 6 Flash
di R. Viscardi
Pagine: 240
Euro: 7,9

Internet Explorer 6 in versione tascabile. Nel consueto formato agile e veloce della collana Flash, tutte le informazioni che servono per scoprire il nuovo browser targato Microsoft, incorporato nel sistema operativo Windows XP



Access 2002 Guida Completa
di Paul Cassel, Craig Eddy, Jon Price
Pagine: 608
Euro: 39,9

Strutturare un database, creare tabelle, report, maschere e query, interfacciarsi con il Web attraverso ASP, conoscere il linguaggio SQL, verificare la sicurezza dei dati.



[Audio e multimedia]

di Lombardo, Valle

Pagine: 408

Euro: 26

Il mondo dell’audio nel contesto più ampio della comunicazione multimediale, dalla rielaborazione del suono al protocollo MIDI. Un testo tecnico per chi studia o lavora nel campo multimediale.